ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Faculty for Engineering Sciences, Computer Science and Psychology
Institute of Databases and Informationsystems

Developing a Complex User Interface for Mobile Data Collection Applications

Bachelor Thesis at Ulm University

Submitted by:
Robin Martin
robin.martin@uni-ulm.de

Reviewer:
Prof. Dr. Manfred Reichert

Supervisor:
Johannes Schobel

2018

Fassung January 22, 2018

Satz: PDF-LaTeX 2$_\varepsilon$

Abstract

The use of paper-based questionnaires for collecting data reveals several downsides, including logistical, cost-related and data quality issues. Despite the increasing digitization and the possibilities evolving from the latter, paper-based questionnaires remained ubiquitous in many application domains. Reasons for this may be insufficient IT knowledge from domain experts, the high development costs for dedicated digital solutions or the lack of domain-specific functionality and ease of use in existing software. In order to solve these issues, the *QuestionSys* framework aims to pursue a digital and easy-to-use approach for collecting data in large-scale scenarios. By providing software solutions for configuring digital questionnaires, executing those questionnaires on mobile devices and evaluating collected results, the framework attempts to support the entire data collection life cycle.

In the context of this thesis, a sophisticated user interface for the *QuestionSys* mobile application was developed. Thereby, an in-depth look at common usability and user interface guidelines for mobile operating systems is taken in this thesis. Further, this thesis presents potential use case scenarios for such an application and their requirements. The user interface is discussed and explained alongside various screenshots of the developed mobile application.

Acknowledgment

At this point, I would like to thank everyone who supported and continuously motivated me during the preparation of this thesis.

Most notably, I would like to thank my supervisor Johannes Schobel for his excellent guidance and support throughout this thesis.

Table of Contents

1

Introduction

Despite the wide dissemination of smart mobile devices over the past decade, the use of paper-based questionnaires to collect data in various application fields, such as psychology and healthcare, remained a common standard [1]. The latter may be caused by a lack of IT knowledge in these fields or the cost intensity of developing sophisticated mobile applications for data collection purposes. Also, digital solutions for data collection might exist, but often lack of general ease of use or certain functionality required by specific domains. However, the traditional paper-based data collection approach goes along with numerous downsides. For example, the use of paper-based questionnaires is more likely to be error-prone, resulting in reduced quality of data [2]. Thereby, errors might occur when filling in a questionnaire (e.g., not following given instructions) or when digitalizing gathered data in time-consuming manual transcription tasks [2]. Also, logistical issues and high costs (e.g., for printing thousands of paper-sheets), especially in large scale studies, must not be ignored [3].

To encounter these downsides, the *QuestionSys* framework, which is currently developed at *Ulm University*, aims to pursue a digital approach, supporting the entire data collection life cycle in an easy-to-use way [4]. Therefore, the framework attempts to provide a set of tools and techniques, empowering domain experts to implement their own “data collection applications” at a high level of abstraction [4]. Via an instrument configurator, questionnaires can be created in the form of generic process-models. The latter can be transferred to a client application, which allows to store and process transferred questionnaires using mobile devices. Data collected using smart mobile devices is then transferred back to a server for evaluation and data analysis purposes.

The aim of this thesis is to create a user interface for the aforementioned client mobile

application. In particular, the interface should comply with common usability standards for mobile operating systems in order to allow for an efficient and at the same time user-friendly way to collect and manage collected data. Furthermore, the resulting user interface should help eliminating some of the downsides, which arise with the use of paper-based questionnaires to collect data.

1.1 Outline

Since this thesis is concerned with the development of an user interface for a mobile application, it mainly deals with aspects regarding usability in mobile applications and mobile application development itself.
To begin with, Chapter 2 gives an overview over fundamental aspects that might be required for further understanding in later parts of the thesis. In Section 2.1, a general introduction to usability and its importance in software applications is given. Further, Section 2.2 introduces *Cross-Platform Mobile Development*, an alternative approach to traditional mobile development strategies. Thereby, the focus is set on *Hybrid Mobile Applications* (Subsection 2.2.1). Following, Chapter 3 is concerned with specific user interface and usability guidelines, which (when followed properly) may lead further towards the goal of achieving high usability in mobile user interfaces. A compilation of guidelines in terms of visual design (Section 3.1), interaction design (Section 3.2) and navigation in mobile applications (Section 3.3) is collected and discussed in this thesis. In Chapter 4, potential use case scenarios for the *QuestionSys* mobile client application are described. From these scenarios, a set of requirements, which need to be fulfilled by the application and its user interface, is derived and presented in Section 4.2. Further, this chapter presents the user interface of the resulting application, which follows the guidelines and requirements elaborated in this thesis, in detail. Finally, a brief summary of aspects covered in this thesis as well as an outlook on how future work regarding the developed application might look like, is given in Chapter 5.

2

Fundamentals

This chapter introduces general aspects, which might be necessary for further understanding in later parts of the thesis. Section 2.1 is concerned with criteria that define usability and gives insights on why usability in general is an important factor in nowadays software applications. Also, an alternative mobile development strategy, the so called *cross-platform* development approach, is presented in Section 2.2.

2.1 Usability

Usability is an important factor in software systems. The term *usability* describes a quality attribute of user interfaces which indicates how easy they are to use. Also, usability is often associated with specific methods to increase the ease-of-use of a user interface during its design and development process [5].
For almost all software applications, such as websites, mobile applications or desktop applications, the usability of the user interface is a key factor for success. If a user interface is not designed to fit the needs of the target user group, it can be hard for them to figure out how to accomplish their desired tasks. Thus, they are more likely to become frustrated and less efficient working with the interface or even stop using the application.
In order to achieve a good and usable user interface, it is necessary to define what makes an interface actually usable. Jakob Nielsen [5], therefore, defined usability by the following five quality criteria:

- **Learnability:** A user interface should be easy to learn. The difficulty of accomplishing basic tasks when first being confronted with a design should be low.

- **Efficiency:** Once a user has learned a design, he should be able to perform tasks within a reasonable amount of time.
- **Memorability:** A user interface should be easy to remember. When returning to a user interface after not using it for a period of time, the effort it takes to relearn it should be minimal.
- **Errors:** A user interface should prevent users from making errors. Further, it should help and make it easy for users to recover from occurring errors.
- **Satisfaction:** A user interface should be pleasant to use.

Fulfilling these aspects as far as possible leads further towards the goal of accomplishing high usability in an application. Nonetheless, developing an interface that fulfills every quality criteria equally is challenging and might not always be possible.
A majority of quality criteria depend on the user's perception, prior knowledge and abilities, which may differ depending on demographic or cultural background. As an example, an interface that is easy to learn for younger people is not necessarily easy to learn for elderly people and vice versa. Also, in some cases it might be the right choice to focus on one quality criteria while neglecting another one as they can stand in direct competition (e.g., Efficiency vs. Satisfaction).
To ensure fulfilling of the quality criteria, guidelines for user interfaces exist that can be applied during the development process. Some of these guidelines are covered in Chapter 3.

2.2 Cross-Platform Mobile Development

Nowadays, market share for mobile operating systems (OS) is divided between Apple's *iOS* and Google's *Android* [6, 7]. For mobile application developers, this means that in order to reach a majority of end-users, it is indispensable to provide an application running on both platforms.
This imposes a massive workload upon developers, since each platform relies on a particular set of specific patterns, rules and guidelines. To comply with platform-specific

standards, applications have to be developed individually by making use of the platform's native programming languages (Java/Kotlin for Android, Objective-C/Swift for iOS), development environments and user interface guidelines.
From the end-user's point of view, the most notable difference lies in the appearance of the user interface. The two platforms make use of their own widgets and patterns to provide the same functionality.
In order to avoid developing two separate platform-specific applications, especially to speed up development time and decrease costs, one can pursue a so called cross-platform development approach. By doing so, the same application is available across multiple platforms. Making use of cross-platform mobile development tools and frameworks, an application can be compiled for multiple target OS from a single code base [8]. Compared to the traditional, native mobile application development, the cross-platform approach offers multiple benefits [8]:

- **Reduction of required skills and knowledge:** Developers are only required to learn programming languages and API's which are provided by the chosen tools and frameworks. They do not have to deal with multiple, platform-specific programming languages or API's of target OS.
- **Reduction of code:** While developing native applications requires to write separate applications for each target OS, the application source code in cross-platform approaches is written once and then compiled for each target OS individually.
- **Reduction of development effort:** The previous mentioned benefits contribute to decrease development time and long term maintenance effort, and, therefore, reduce the overall development costs.

2.2.1 Hybrid Mobile Applications

There exist different types of developing such mobile applications that follow the cross-platform approach. One of them is the hybrid mobile application. As the name suggests, hybrid mobile applications are some kind of crossover between native applications and web applications, including benefits from both sides. Web applications, in turn, run in a

web browser, thus, have no platform dependencies. However, web applications have limited access to a mobile device's native functionality (e.g., sensors, services or inputs). Hybrid applications, just like native applications, are able to access the underlying device hardware [9]. Further, hybrid mobile applications can be distributed to and downloaded from the platform-specific application marketplaces such as the *App Store* or the *Google Play Store*.

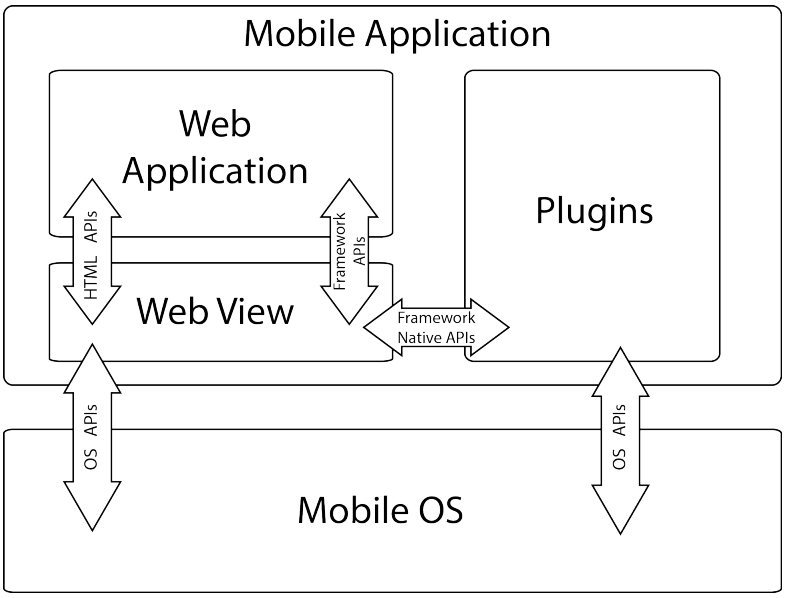

Figure 2.1: Typical software architecture in hybrid mobile applications [10]

As Figure 2.1 shows, hybrid mobile applications consist of three major components [10]:

- **Web Application :** Typically, hybrid mobile applications are implemented as web applications. While application logic is written in JavaScript, the user interface relies on *Hypertext Markup Language* (HTML) and *Cascading Stylesheets* (CSS).
- **Web View :** The Web View can be seen as a slimmed down version of a mobile device's web browser. It is a native platform-specific component which provides a run-time environment for the application.
- **Plugins :** Due to the restricted access to device resources and functionality, hybrid applications make use of dedicated plugins to access OS specific resources. Via

foreign function interfaces[1], platform-specific native code can be invoked using framework-specific JavaScript API calls.

To develop hybrid mobile applications, one can fall back on several development frameworks. Those frameworks provide developers with all kinds of helpful tools and building blocks to minimize development effort as well as to give the resulting application a native, platform-specific look and feel using web technologies only. In Section 4.8, a more detailed presentation of one hybrid development framework, namely the *Ionic* framework, is given.

[1] A mechanism for applications written in one programming language to call routines or services written in a foreign programming language

3

User Interface Guidelines

Developing user interfaces is a complex procedure, as various aspects have to be considered in order to make it a good user interface. These aspects include, for example, the actual device displaying the user interface, the OS running on the device, the context an application is used in, and most importantly, the end-user, who is going to interact with the user interface.
While in Section 2.1, quality criteria defining the usability of interfaces were presented, this chapter is concerned with specific rules and guidelines leading to the fulfillment of those criteria. In detail, these rules and guidelines are looked at in the context of mobile applications, especially applications for the two major mobile OS, namely *iOS* and *Android*. As already mentioned in Section 2.2, the user interface of mobile applications for these platforms can differ greatly. Hence, their platform-specific user interface design and usability guidelines, the *iOS Human Interface Guidelines* and the *Material Design Guidelines*, are taken into consideration. Furthermore, principles and recommendations from literature dealing with user interface design and usability in the context of mobile applications and websites (as certain aspects are applicable to both), are summarized. The objective of this chapter is to specify a set of general user interface and usability guidelines that are applicable to mobile applications in the context of data collection.

3.1 Visual Design

In user interfaces, visual design refers to the way content is visually presented. Thereby, different minor aspects, such as the use of colors or the choice of suitable typefaces, contribute to an overall visual design. Further, a well thought visual design can have a

positive effect on usability in general, as for example an interface which is pleasant to watch may also be pleasant to use.

3.1.1 Color & Contrast

In human-computer interaction, color plays a critical role, and, therefore, it does in user interface design. Color can draw attention to certain interface elements, communicate information by helping users to understand and interpret the application's content, as well as influence the user's emotions and actions [11]. By applying colors in the right way, the latter may increase usability of an interface dramatically, whereas, when applied wrongly, they can act as a hindrance to usability.
A broad rule of thumb is, to use color sparingly. Overusing colors in user interfaces is often perceived as distracting and fatiguing the human eyes. Since color is also used to communicate information and indicate the importance of content displayed, using too many colors leads to a decrease of importance of communicated information [12]. In other words, using less color increases its ability to call attention to important information or elements. Taking these aspects into consideration, *iOS* and *Android* usability guidelines suggest to choose one primary color, that is dominant throughout the overall user interface [12, 13]. To guide the user through the interface, highlighting and drawing attention to specific user interface elements, one can either colorize those elements in different gradations of the primary color, or use a secondary (complementary) color to set accents. For example, accent colors can be used to highlight interactive elements [12, 13], such as buttons, text input fields or sliders.
Further aspects have to be considered, when using color to communicate the state of certain interface elements, for example to differentiate between enabled and disabled interactive elements (e.g., buttons). First, the perceived meaning of certain colors can differ depending on a user's cultural background [12]. While, for example, in some cultures, the color *green* has a positive connotation (e.g., success) and the color *red* a negative one (e.g., error), the same colors are associated with opposite connotations in other cultures. This can mislead users, to wrongly interpreting the meaning of elements the respective color is applied to.

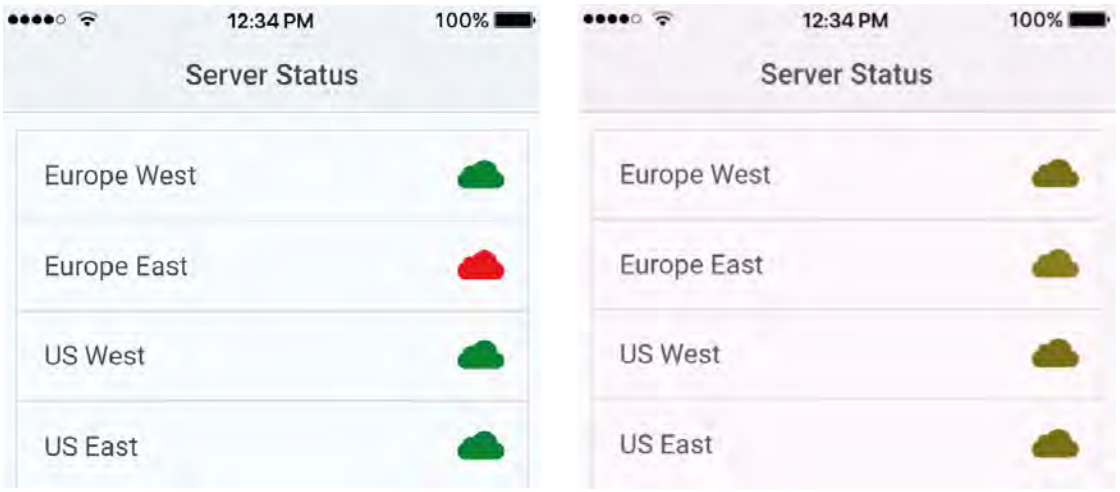

Figure 3.1: Normal Vision vs Colorblind Vision

Also, people with visual impairments have to be taken into account, when selecting colors for user interfaces. For instance, approximately 1 out of 12 men (8%) and 1 out of 200 women (0.5%), worldwide, are affected by some kind of colorblindness [14], making it hard for them to discriminate certain colors (mostly red-green or blue-yellow color combinations [12, 15]). Further, due to physiological changes of aging, the visual perception of elderly people can become restricted, making color-combinations that are clearly distinguishable for young people indistinguishable for the elderly [16, 15]. In Figure 3.1, potential effects of color selection for visually impaired people is visualized. To avoid culture related and constitutional misunderstandings regarding color, one has to make sure selected colors in a user interface are sending the appropriate message and selected colors are clearly distinguishable by everyone. Furthermore, instead of relying solely on color to communicate information or the state of certain elements (as it is done in Figure 3.1), using different color-shape or color-text combinations can help prevent previously mentioned misunderstandings [12].

Since tablets and smart phones are mobile devices, their users are not bound to a certain place to use them, compared to, for example, PC users. This environmental complexity, as well as limitations in visual perception of certain user groups, can lead to serious issues regarding visibility and readability of displayed content in mobile user interfaces. To ensure, that the mobile application's content is clearly visible and accessible for every

user, under various conditions (e.g., lighting conditions outside), the contrast between the content color and the color of the background it is displayed on should be as high as possible. According to the *World Wide Web Consortium* [17], large text (e.g., headlines) should have a contrast ratio of at least 3:1 against their background, whereas the contrast ratio for normal sized text should be 4.5:1, at least. The same contrast ratios should also be considered for displayed icons [18]. Such contrast ratios can be achieved by using light colors for text on dark background and vice versa.
In Table 3.1, a brief summary of user interface guidelines discussed in this section is presented.

ID	Guideline
COL1	Use color judiciously
COL2	Use one primary color throughout the application and add a secondary color to highlight important (interactive) elements
COL3	Avoid using color as standalone indicator to communicate the state of an element
COL4	Avoid red-green and blue-yellow color combinations
CON1	Provide an adequate contrast ratio of at least 4.5:1 for normal sized and 3:1 for large text and icons

Table 3.1: User interface guidelines for color and contrast

3.1.2 Typeface

User interface design is mostly about communicating information towards the user. In mobile applications, the most commonly used communication form is through the display of textual content [19]. But there are substantial differences between reading text from a display and reading text from paper. Reading text from a computer screen for example takes about 25 percent longer than reading text from paper [19]. Regarding the limited screen sizes and environmental complexity of mobile devices, they are more likely to decrease the user's ability to read text even further [19]. While in some contexts, the

loss of information due to bad readability does not lead to further problems, the loss of information in the context of data collection scenarios is critical, since it can affect the quality of collected data dramatically. Hence, to avoid readability issues, selecting an appropriate typeface for displaying textual content is necessary.

By default, the typeface used in iOS applications is *San Francisco* [20], while most Android applications rely on *Roboto* as their standard typeface [21]. Both are sans-serif typefaces, which were invented specifically to be displayed in user interfaces. Although the quality of screens increased over the last years, enabling the proper display of serif typefaces, the use of serif typefaces in mobile user interfaces tends to impose readability problems upon certain user groups dealing with visual impairments (e.g., elderly people [16]). To avoid such issues, using sans-serif typefaces (e.g., *Roboto*, *San Francisco* or *Neue Helvetica*) in mobile user interfaces might be the best choice.

To emphasize the importance of certain words or phrases, font type variations can be applied [20]. For example the weight-variations of types can be used to reflect relative importance of words the variation is applied to, compared to other texts. Therefore, page titles and headlines should always have increased type weight, to match their importance. However, type styles should be applied conservatively, as applying too many variations and styles can, in turn, distract users and therefore hinder readability [21, 16].

Finally, longer text phrases should be displayed including capital, as well as, lower case letters. Reading paragraphs containing capital letters only can slow down reading speed by 10 percent [19], as the shape of certain words is no longer recognizable for the user and the text has to be read word by word and letter by letter.

Table 3.2 is sums up the most important points, concerning the use of typography in mobile user interfaces.

ID	Guideline
TYP1	Use sans-serif instead of serif typefaces
TYP2	Emphasize important content by applying type variations
TYP3	Apply type variations conservatively
TYP4	Avoid using only uppercase letters in longer texts

Table 3.2: User interface guidelines for typography

3.1.3 Iconography

When designing user interfaces, relying on icons may bring numerous benefits. Icons, by definition, are small, pictorial images, which are used to represent certain objects, actions or ideas [22, 23], often in a metaphorical and abstract way. The type of icons can, again, be subdivided into two groups [24]:

- **Product icons** are portrayals of the services, tools and products a brand is providing, whereas
- **System icons** are system specific representations of objects (e.g., a file or directory), commands (e.g., *backwards arrow* to return to previous page) and feasible actions (e.g., plus (+) sign to add items)

Especially when designing interfaces for mobile applications, icons can unfold their full potential. Their relatively small size allows multiple icons to be displayed in size limited display areas (e.g., Tab Bars) while still being big enough to act as touch targets. Further, when designed properly, icons are easy to recognize and easy to remember [22], which can affect usability criteria such as learnability, memorability and efficiency in a positive way. When selecting icons for a user interface, several aspects have to be considered in order to make them work the intended way.
To begin with, understanding of icons is often based on a user's prior experience from other applications as well as the context the icon is used in [22]. In other words, the recognition and understanding of the meaning of icons is faster, if users are already familiar with certain icons, or, if the meaning of an icon in a specific context is self-explanatory. Accordingly, relying on icons from platform-specific icon sets (where possible) is highly suggested. Users are more likely to be familiar with the meaning of such icons, since the latter are used in other applications across the platform as well. Nonetheless, one should not apply platform-specific icons for the single purpose of applying them. If there is no platform-specific icon representing an intended meaning or beahviour, it should not be applied to represent that meaning, since this can lead to misunderstanding. In such cases, one can decide to design specific icons representing the intended meaning best [25]. Doing so, platform-specific design standards should be applied to remain aesthetic

consistency.
To avoid icons being misinterpreted by users, as well as to help users familiarizing with the user interface, they should always go with a small, textual label [22, 25]. The label should be displayed either at the bottom or besides the icon itself and contain the name of the action that corresponds with the icon.
In spite of everything, when selecting icons for a user interface one should always consider the *five second rule*. The rule states, that “if it takes [...] more than 5 seconds to think of an appropriate icon for something, it is unlikely that an icon can effectively communicate that meaning” [22].
A compilation of relevant guidelines concerned with the utilization of icons in mobile user interfaces is presented in Table 3.3.

ID	Guideline
ICO1	Avoid using icons, if it takes too long to think of an appropriate icon
ICO2	Use icons in size limited display areas over plain text labels
ICO3	Make use of platform-specific icon sets
ICO4	Apply icons in a consistent way
ICO5	Provide small textual labels at the bottom or besides icons

Table 3.3: User interface guidelines for icons

3.1.4 Terminology

Every word and phrase used in a user interface is part of a conversation between the user and the system. Therefore, the system should speak the users’ language to make them feel comfortable and build trust in the system. This means, that rather than using sophisticated, system-oriented wording, the language should be simple and user oriented [26]. Technical or domain-specific jargon might of course be understood by domain experts, but in the same way can intimidate a broader user audience [27].
Since users should be comfortable in using an application, the general tone of language

should be a user-centered, inviting and positive one. To achieve this, it is suggested to speak in a second person conversational style, addressing the user directly using pronouns such as “you” or “your”. Likewise, the pronoun “we” should be avoided in most cases as it shifts the focus from *what the user can do with an application* towards *what the application can do for the user*, which might be perceived as insulting or patronizing [27, 28].
When it comes to interactive elements, users should immediately know, which action a certain element invokes. This can either be achieved by applying appropriate icons, as mentioned in Subsection 3.1.3, or by providing text labels for interactive elements. These labels, however, should contain words that describe the action that is invoked, using action verbs such as “download”, “upload” or “send” [27].
Table 3.4 briefly summarizes the most important guidelines from this section.

ID	Guideline
TER1	Keep words and phrases simple and informative
TER2	Keep the tone of language polite, positive and user-centered
TER3	Avoid technical or domain specific jargon
TER4	Write phrases in second person conversational style, avoid first person
TER5	Use action verbs for labeling interactive elements

Table 3.4: User interface guidelines for terminology

3.2 Interaction Design

While Section 3.1 was mostly concerned with the visual presentation of content in mobile applications, this section covers different aspects regarding the interaction between the user and the application’s content. In detail, an overview of guidelines, enabling users to perform tasks in a mobile application efficiently and with ease, are presented.

3.2.1 Gestures

Before the widespread of touchscreen displays in mobile devices, physical buttons or pointing devices (e.g., mouse) were the standard way of interacting with content on user interfaces. Nowadays, in turn, a touchscreen display covers almost the entire front-side of a mobile device, leaving no space for physical buttons. Also, pointing devices are inconvenient to carry around and might get lost easy. Hence, touchscreen devices rely heavily on gestural controls, using the human hands to interact with on-screen content. Although there are many gestures relying on the mobile devices sensors (e.g., Accelerometer or Proximity sensor), the main focus of this section are touch-based gestures. A set of standard gestures, provided by nearly every mobile OS is presented in Figure 3.2.

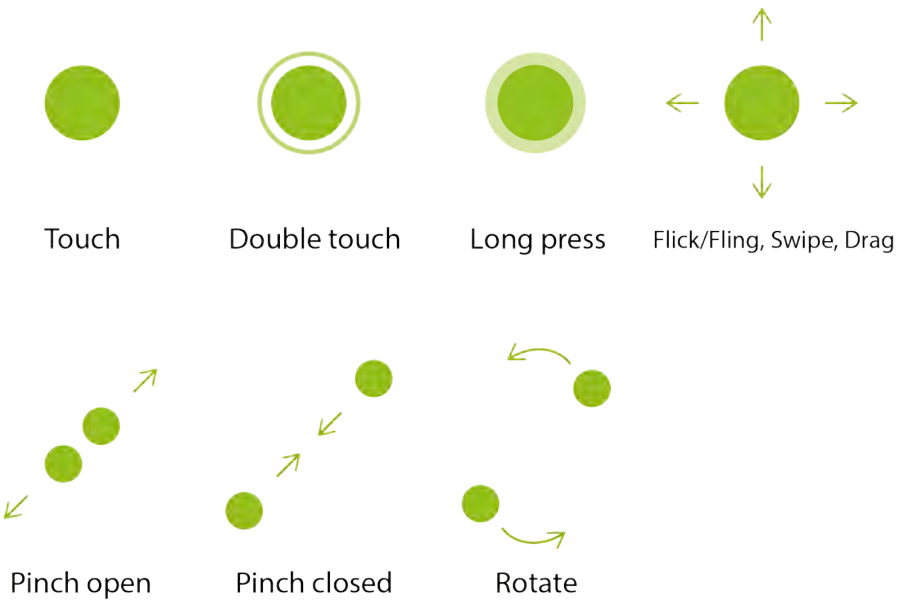

Figure 3.2: Standard touch gestures for multitouch enabled devices [29, 30]

While mobile OS use almost the same gesture patterns, the outcome of the gestures can differ between platforms. To clarify, Table 3.5 gives a brief overview of actions associated with standard touch gestures on *iOS* and *Android*. However, the gestures presented in Figure 3.2 and Table 3.5 only cover a small amount of possible gestures for multitouch

Gesture	iOS	Android
Touch / Tap	Activates screen element	
		Canceles / escapes current task (dialog / menu)
Double touch / tap	Zooms in / Zooms out	
Long press	Displays view for cursor positioning	
	Enters a mode allowing items to be rearranged	Enters a mode allowing items to be selected
Pinch	Zooms in / Zooms out	
		Expands / Collapses content
Drag	Moves element, scrolls / pans precisely	
Flick / Fling	Scrolls / Pans quickly	
Swipe	Reveals off-screen content, Reveals hidden elements, Switch between in-content views, Refreshes content	
	Returns to previous view	

Table 3.5: Actions associated with standard gestures on iOS and Android [29, 30]

enabled devices. Gestures can appear in more complex forms, for example, by combining standard gestures or relying on alternative patterns (e.g., drawing symbols on screen to perform certain actions). The latter allows to have various additional functionality, while saving screen real estate. Nonetheless, the more complex a gesture gets, the harder it gets to discover, learn and remember, as there is no on-screen reminder indicating that a certain gesture can be performed [31]. Therefore, it is suggested to rely on standard gestures. The chances are high that the user is already familiar with these gestures from prior experiences with other applications from the same platform [29], resulting in reduced additional learning effort for the user.

Further, when making use of standard gestures, one should apply them to be used in the intended way (see. Table 3.5), defined by the corresponding platform. Using standard gestures to perform non-standard actions can lead to confusion and, therefore, should

be avoided [29]. Since there is no visual indicator for gestures one can not expect a user to know that a gesture can be performed in a certain context. When possible, there should exist visual shortcuts to supplement these gestures [29]. As an example, to supplement a “long press” – gesture in a *list view*, which enters a selection mode, a “select” – button could be used in the same view, to provide the same action in a more discoverable way.
Finally, due to the nature of large touchscreens on mobile devices, people often activate certain gestures by mistake (e.g., accidental button click) [31]. The effects of accidentally triggering a gesture can range from harmless (e.g., accidental scroll down) to critical (e.g., accidental deletion). To prevent critical outcomes when activating gestures by mistake, one has to make sure that gestures are reversible, for example, by presenting confirmation alerts.
The rules and guidelines regarding gestures are summarized in Table 3.6.

ID	Guideline
GES1	Avoid using too complex gestures, rely on standard gestures
GES2	Avoid using standard gestures for non-standard actions
GES3	Provide discoverable shortcuts to supplement gestures
GES4	Make gestures reversible

Table 3.6: User interface guidelines for gestures

3.2.2 Data Entry

In many mobile applications, entering data is an essential form of interaction. Especially for applications in which data must be acquired from user inputs and then be processed accordingly, this form of interaction comes to focus. Mobile applications for data collection purposes, thereby, embody the upper extremity, as their main intent is to gather and process data entered by users. Hence, it is from great importance to design data entry interactions to be pleasant and efficient to be performed by the user, on the one hand. On the other, the validity of entered data may be ensured by diminishing incorrect inputs.

To accomplish data input tasks on touchscreen devices, most mobile OS provide virtual, on-screen keyboards. While virtual keyboards eliminate the necessity to carry around physical keyboards, multiple problems can arise when making use of them. The key sizes of virtual keyboards depend on the dimensions of the display. As a result, the key size of virtual keyboards is often smaller compared to physical keyboards, which can lead to a loss in typing efficiency, especially when entering longer texts [32]. In general, typing efficiency will increase with frequent usage of virtual keyboards, as it is heavily based on practice. In this context, especially novices will have problems in finishing data input tasks in an adequate time. Also, motor impairments (e.g., decreasing pointing ability) can affect typing efficiency in a negative way. To avoid data entry inefficiency, input methods requiring a virtual keyboard should be kept to a minimum. Rather than using text fields to acquire data, data entry processes can be simplified and accelerated by relying on alternative input methods (e.g., *date picker*, *dropdown fields*, *radio button groups*) [33]. The latter allow users to choose from a predefined set of available options instead of typing in a response, making it more comfortable for a user to input data and reducing errors in resulting data (e.g., wrong date format). Further, by pre-filling input fields with data that can be automatically gathered from the system or by providing reasonable default values, the amount of typing a user has to perform can be decreased dramatically [33, 34].

However, sometimes the usage of text input fields is inevitable, for example, when options for an input can not be predefined (e.g., textual descriptions) or when the amount of available options for an input is from such height, that looking for one specific option would take much longer, than just typing the respective value. In such cases, steps have to be taken in order to minimize the effort it takes for a user to enter data.

To begin with, input fields should always go with a label or placeholder, indicating the purpose of the input it is applied to [33]. Additionally, applying prefixes or suffixes to input fields can help to put them in a certain context, by, for example, clarifying the unit of numerical inputs [35]. To make input fields visually stand out from surrounding content and, therefore, making it easier for users to identify them as such, one can apply a transparent rectangular fill, which encloses an input label or text [35].

Another possibility to minimize the input effort for a user, is to display different keyboard

layouts according to the type of an input [34]. Mobile OS mostly provide multiple keyboard layouts by default, making characters, which are often used in a certain context (e.g., *At(@)-sign* for e-mail inputs or a set of numbers for numerical inputs), easier to access. Using appropriate keyboard layouts diminishes the number of steps it takes for a user to reach desired characters, and can improve input efficiency dramatically.
Lastly, a user should always be aware of the current state of an input field. Not evaluating input fields until a user tries to proceed, can easily lead to frustration, as in the worst case, multiple inputs have to be revised. Therefore, input fields should be validated dynamically, giving the user an instant feedback about the validity of a given input [33]. If a user input does not match certain constraints imposed for its validity (e.g., exceeding maximum length of text input), the user should be notified by, for example, displaying an error message with useful instructions on how to change the input to be valid [35].
Table 3.7 briefly summarizes the most important guidelines described in this chapter.

ID	Guideline
DAT1	Keep data entry tasks requiring keyboard input to a minimum
DAT2	Provide alternative input forms enabling users to choose from a set of available options
DAT3	Prepopulate input fields where possible
DAT4	Make input fields easily discoverable
DAT4	Provide labels and placeholders to communicate the purpose of an input field
DAT5	Display suitable keyboard layouts for different input types
DAT6	Always communicate the current state of an input field

Table 3.7: User interface guidelines for data entry

3.3 Navigation

The limited display size of mobile devices is making it hard to display many information in a single view, while still remaining clarity and structure. As a result, information is

often organized in hierarchical structures throughout the application. In order to reach certain information or application functionality, the user has to *navigate* through different views of an application. Thus, the question arises how to implement such a navigation, so that users can navigate through an application in an intuitive and easy way.

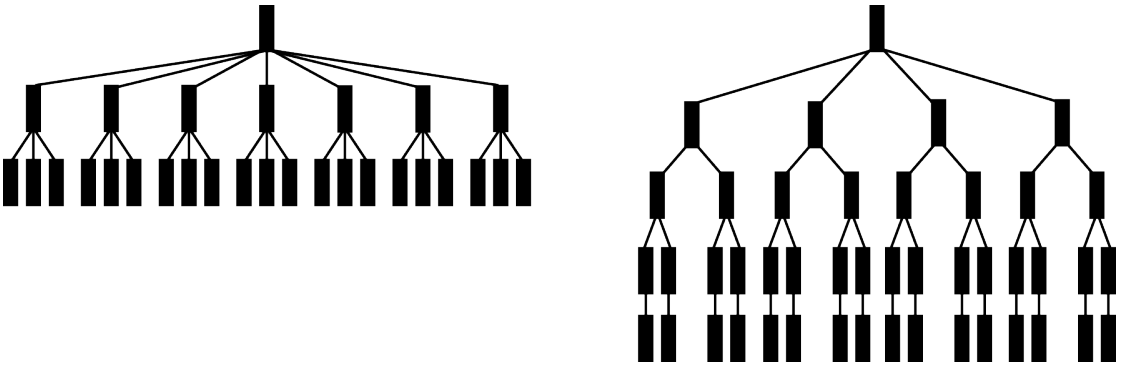

Figure 3.3: Flat structure (left) vs deep structure (right)

There are two main types of navigation structure used in mobile applications and websites (see Figure 3.3), flat and deep navigation structures [36, 37]:

- **Flat navigation structure:** Content is logically grouped into categories, where each category has its own top level view. To navigate between categories, the user can easily switch between those top level views.
- **Deep navigation structure:** One choice per screen has to be taken in order to navigate to the next deeper hierarchy level, until a certain endpoint (c.f., Figure 3.3, leafs in tree) is reached. To navigate to another destination, either every step taken has to be retraced (e.g., via *Breadcrumb* navigations), or navigation has to be restarted from the root view.

Often, these structures do not appear in their initial form, but rather in a mixture of both. Thereby, in applications with a flat navigation structure, each category can have multiple sub levels.
The application should support users in reaching desired information or functionality as easy and fast as possible. Therefore, one has to figure out first, how important certain content, information or functionality is, from the user's point of view, and how often it is needed on a regular basis [38]. Important information and main functionality then should

be placed higher regarding the navigation hierarchy, enabling users to reach it faster. In turn, side functionality and less important information can be nested in deeper levels of the hierarchy. In general, it is suggested to keep hierarchies as flat as possible, since the deeper content is located, the less discoverable it becomes for the user [36].
Regardless of the application's underlying navigation structure, users should be aware of their current location inside the application at any time [37, 39]. Otherwise, not only new users, but also those not using the application on a regular basis, can get lost and, as a result, work less efficient with the application. To achieve location awareness, pages should be titled consistently, indicating the current location. Further, menus for navigating, (e.g., Tab Bar or Navigation Drawer) highlighting the user's current position, can be used. Additionally, when navigating to sub level views, one should always be able to return to a previous view, located higher in the navigation hierarchy. To achieve this, the different platforms rely on various patterns. On *iOS* devices for example, going back to a parent view can be implemented as a swipe gesture [29]. Though, this gesture is hard to discover for users that are not familiar with the platform, as there is no visual indicator. On *Android* devices, one can use the hard- / software back button to return to previous views. While this pattern is easy to discover (as the button is always visible), it is not solely used for screen-to-screen navigation in an application. It also supports other behaviors (e.g., dismissing floating windows, contextual action bars or returning to the home screen from a root view) [40]. One commonly used pattern, across platforms, for returning to a parent view from a child view is a dedicated button in a top left position on nested views ("Up"-Button on *Android*, "Back"-Button on *iOS*). In this position, it is easy to discover for the user and provides predictable behavior throughout the application.
The user interface guidelines derived from this chapter are summarized in Table 3.8.

ID	Guideline
NAV1	Implement navigation in a way that supports the user in reaching desired content or functionality with ease
NAV2	Avoid the navigation structure to become too deep
NAV3	Always indicate the user's current location inside the application
NAV4	Always equip sub level views with a "Back"/"Up" button to indicate the possibility to return to its parent view

Table 3.8: User interface guidelines for navigation

4

Application Scenario

Taking the guidelines, gathered in Chapter 3 into account, a mobile application for data collection purposes was developed. The target platforms, therby, were *iOS* and *Android*. This chapter gives insights into the development process of the aforementioned application's user interface. In the first place, potential use case scenarios for such an application are circumscribed. Next, requirements to the application and its user interface, that can be derived from these circumscriptions are defined. Finally, a presentation of actual design decisions that were made to satisfy requirements, using guidelines from the previous chapter, is given.

4.1 Use Case Scenarios

By now, the main focus of the *QuestionSys* – Framework are the clinical and psychological domains. However, the range of use cases is not limited to the latter. Besides application in scientific research, the system could also find usage in institutional or industrial fields.

One real-world application scenario, where digital questionnaires were used to conduct a trial in the field of clinical psychology, is described in [3]. The intention of this trial, which took place in rural areas in *Burundi*, was to investigate the *Post-Traumatic Stress Disorder* (PTSD) of ex-combatants and soldiers. Due to logistical issues (i.e., transporting large numbers of paper-based questionnaires) and governmental impositions (regarding privacy and security issues), an electronic questionnaire application was developed. Using this application, the international team of psychologists was able to fill in the

questionnaire, during an interview with a participant, in about 2-3 hours. Corresponding results were encrypted, stored on the device and could later be decrypted and evaluated. In [41], multiple potential application scenarios for the *QuestionSys* client application were described. The scenario *application for asylum*, thereby describes how the application could be used to facilitate and speed up the application process for asylum seekers, as well as to diminish operating expenses for the responsible institutions. The motivation behind this scenario is the rising number of refugees, seeking for asylum. Often, responsible offices are unable to cope with the large number of applications, which can lead to several months of processing time until a final decision can be made. Institutions could, therefore, provide tablets with the *QuestionSys* client application, enabling refugees to fill out the application themselves. Furthermore, it would be possible, to give a first assessment about the success of the application, based on given answers.
Another related work [42] dealing with application scenarios for digital questionnaires describes the usage of the latter for POS (*Point of Sale/Service*)-Surveys. In POS-Surveys, consumers of certain products or services, a business provides, are interviewed directly at the place of purchase about relevant parameters concerning the provided product or service. For businesses, such surveys can be a major indicator for customer satisfaction and can further help increasing product and service quality. The corresponding use case in [42], describes a hotel business, using paper-based questionnaires to measure their guests satisfaction with provided services in order to continuously increase service quality. With hotel guests decreasingly paying attention to such questionnaires, the hotel decides to exchange paper-based with digital questionnaires in order to make the fill-in process more pleasant for their guests and as a result have more guests to fill-in questionnaires. By providing smart-devices, guests could make use of on-board sensors (e.g., camera or microphone), for example, to capture defects or shortcomings. This would, in turn, enable hotel personal to identify and remedy occurring issues more effectively and, therefore, increase service quality.

4.2 Requirements

Considering the versatile fields of application, including those mentioned in Section 4.1, multiple quantitative as well as qualitative requirements, that the mobile application has to fulfill in order to be suitable for each possible scenario, arise.

- A user should be able to **log in** to the application with an existing ***QuestionSys*-Account**.
- A user should be able to **download** and **execute questionnaires** linked with his ***QuestionSys*-Account**.
- Due to potentially long lasting interview sessions, a user should be able to **pause** and later **resume** to a **questionnaire instance**.
- Collected **results** should be **stored locally**.
- A user should be able to **manage** and **view** collected **results**.
- A user should be able to **upload** collected **results** to the ***QuestionSys*-Server**.
- A user should be able to **modify** relevant application **settings**.
- The application interface should be **designed** in a **neutral** way in order to be used across various domains.
- The user interface of the application should be **self-explanatory**.
- The fill-in process of a questionnaire should be **pleasant** and **time-efficient**, as often users participation in surveys is voluntary.
- The application should allow for **customizations** in the user interface, to fit the user's needs and to enable application providers to support their branding.
- The application should support **multiple questionnaire modes** (e.g., interview or self-rating).
- The application should support **multiple languages**.

4.3 Application Structure

The basic structure of the application (see. Figure 4.1) is a result of analyzing functional requirements described in Section 4.2. The application is divided into two main parts, a public area (*green*) and an area for administration purposes (*orange*), which requires further authentication to access. Regarding the requirement for different questionnaire modes, this split allows to fill in questionnaires in self-rating mode, with unauthorized users not being able to manipulate or have insight into already collected data. When being in the *public area*, a user is only able to execute specific questionnaires and view the application's imprint, whereas accessing the *administration area* makes further application functionality available for the user. In this area, application content is logically

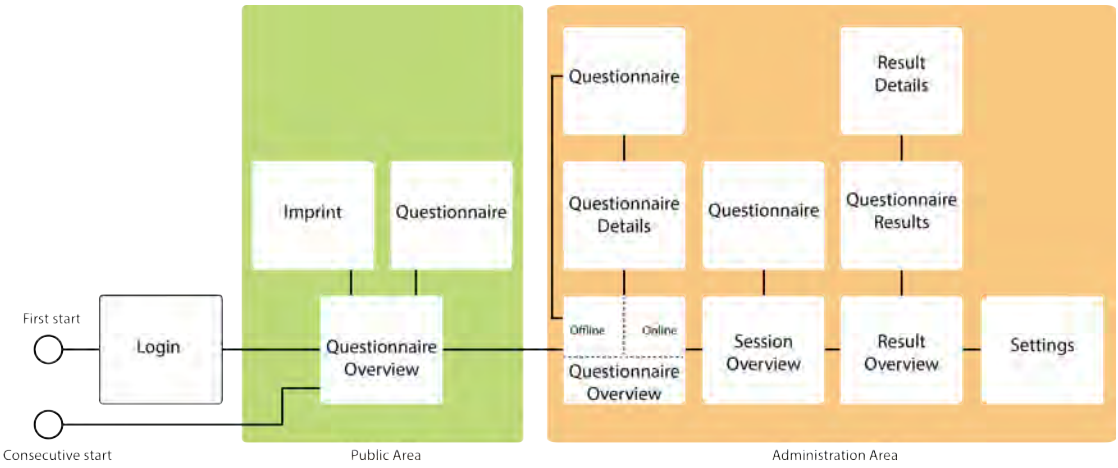

Figure 4.1: Abstract representation of the application structure

grouped according to the specified requirements. The application provides *top level views* for managing questionnaires and their collected results, overview and resuming questionnaire instances as well as one for configuring settings. As Figure 4.1 shows, some of the *top level views* have *nested views* to provide more detailed information about, for example, specific questionnaires or results. Due to the logical grouping, the resulting navigation structure of the application is rather flat than deep (complying with **NAV2**), with a maximum of two *nested views* per *top level view*.

The structure and design of the views themselves, as well as the navigation between them is presented in the following sections.

4.4 Visual Design

During the development of the user interface, the guidelines from Chapter 3, regarding the visual design in mobile applications, were taken into consideration. Most of these guidelines were applied globally (e.g., guidelines concerned with color) to remain visual consistency throughout the application. Insights on how the guidelines were implemented in detail, are given in the following sections.

4.4.1 Colors

Regarding the requirement to empower users to customize the application interface, a set of predefined color schemes is provided (see Figure 4.2). These themes can be switched according to personal preferences. Thereby, the themes mainly differ in terms of their primary and secondary color. Throughout the user interface, the primary colors were used to colorize the *header bar* as well as interactive elements (e.g., buttons) , in order to give each page a consistent look (**COL2**).

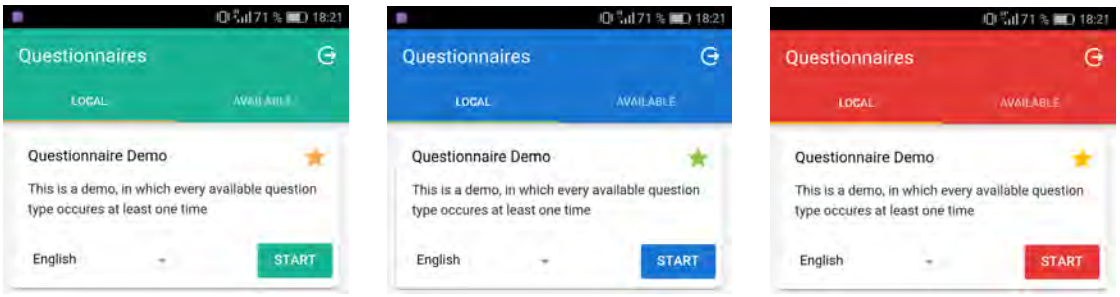

Figure 4.2: Overview of available color schemes

Further, they are partially used to supplement state indications of certain interface elements. However, in conflicting cases (e.g., indicating the currently active *header bar* segment) the primary color is replaced by a secondary color (**COL2**). To avoid color related misunderstandings (e.g., for people having problems discriminating certain colors), colors are never used as a standalone indicator to communicate an element's state (**COL3**). Such indications are always combined with a second characteristic (e.g.,

different shape or text). To comply with **COL4** it was taken care that no red-green and blue-yellow color combinations exist between chosen primary and secondary colors.
In order to ensure proper contrast ratios between textual content and its background (**CON1**), and, therefore, increase readability, text color is mostly set to black (or gradations of black). In turn, the background color remains white (highest possible contrast ratio *21:1*). For textual content on colorized backgrounds, text color suggestions from the *Material Palette* [43] were taken into account.
Further insights on how color is applied inside the application's user interface can be taken from the screenshots in the following sections.

4.4.2 Typeface

Throughout the application, textual content is displayed using sans-serif typefaces (**TYP1**). Furthermore, when running on *Android* devices, the default typeface *Roboto* is applied to comply with platform standards. In turn, when running on *iOS* devices, textual content is displayed using the standard *San Francisco* typeface.
Regarding **TYP4**, most of the application's textual content consists of mixed case letters, with only one exception. The *Material Design* style-guide suggests for button texts to be capitalized. Accordingly, when running on *Android* devices, text on buttons is displayed using capital letters only.

4.4.3 Iconography

The application often relies on icons in order to save screen space (**ICO2**), for example inside the *tab* or *header bar*, as well as for esthetic purposes. Furthermore, according to the underlying mobile OS, platform-specific icon sets were used (**ICO3**). This way, the same icon might have a slightly different appearance on *Android* compared to *iOS*. To remain consistency inside the application as well as across the respective platform, it was tried to use icons for the purpose they were designed for (e.g., "gearwheel"-icon for settings or "trash bin"-icon for deletion). Moreover, the same icon is never used to

communicate different actions in another context (**ICO4**). Since the application context might be fairly specific, regular icon sets do not contain icons, which communicate the intended meaning for questionnaire related objects (e.g., paused instances or results). Hence, the icons depicted in Figure 4.3 were added to the system icon set of the respective platform. For actions or objects, which were hard to visualize as icons (e.g., local or available questionnaires), icons were forgone and replaced with text labels (**ICO1**). In cases, where the meaning of an icon might not be self explanatory, but its understanding is critical for the user (e.g., tab bar icons used for navigation purposes), icons are supplemented with small textual labels indicating their intended purpose (**ICO5**). The screenshots of the application in the following sections give further insights on how (and where) icons were applied.

Figure 4.3: Icons for Questionnaires, Sessions and Results

4.4.4 Terminology

Since fields of appliance for the application may go further than scientific research, it was tried to keep the amount of domain specific jargon to a minimum (**TER3**). Words and phrases in dialogues with the user (e.g., confirmation alerts) are kept short and informative using a general, polite and positive tone of language (**TER1** & **TER2**). By writing phrases in second person conversational style (**TER4**), a more user-centered tone is achieved. Further, interactive elements (e.g., buttons) are either labeled with an icon communicating the meaning of an action which is triggered or with an action

verb (**TER5**) describing the triggered action (e.g., “DELETE”). Example dialogues are presented in Figure 4.4.

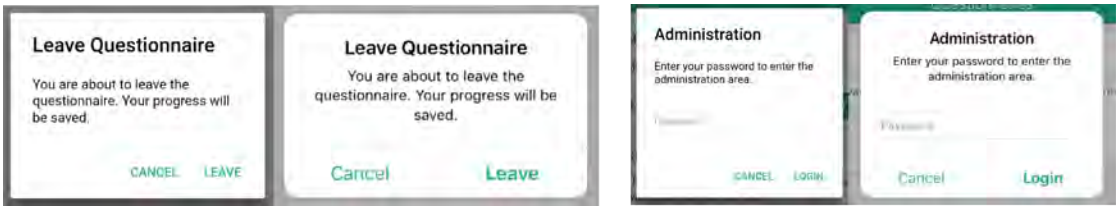

Figure 4.4: Dialogues for leaving questionnaire execution (left) and entering the administration area (right)

4.5 Questionnaire Interface

This section focuses on the interface, which is displayed when executing a questionnaire instance. While the actual content of questionnaires might differ, each questionnaire consists of a set of pages. Each page, in turn, is a composition of different elements (e.g., headlines, texts or questions). Figure 4.5, thereby, shows the representation of a single questionnaire page inside the application. To begin with, a fixed *header bar* resides on top of the screen. The bar itself contains the title of the questionnaire, which is executed at the moment, as well as an *“Up/Back”*-button, which allows the user to exit the questionnaire instance. The *header bar* is contained by every single page of a respective questionnaire, making users not only able to stop the execution of a questionnaire at any given point of time, but also aware of the questionnaire they are filling in at the moment. To avoid users accidentally stopping the questionnaire execution, a confirmation alert is displayed when trying to exit (**GES4**). Underneath the *header bar* is where the actual content of a questionnaire page, namely the set of questions, resides. Questions are displayed one above the other, with each question always taking up the full display width. The height of a question element, however, is calculated dynamically according to the respective question’s content. As Figure 4.5 shows, the content of a question element is divided up into two main parts, a descriptive part and an interactive part. For proper

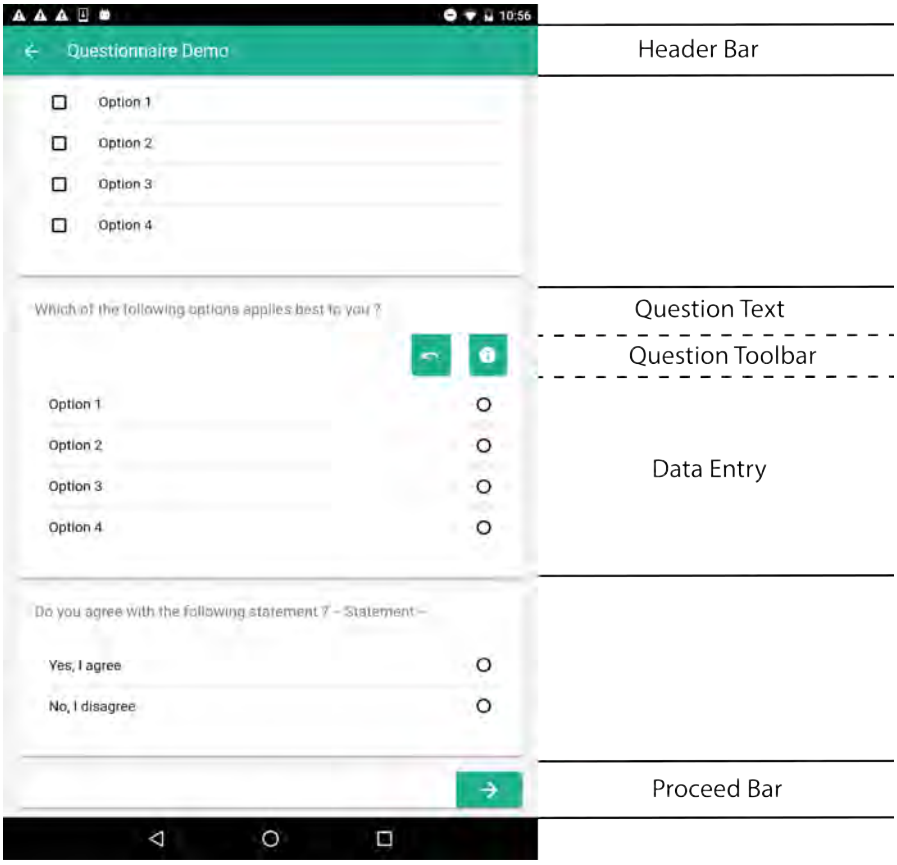

Figure 4.5: Basic Structure of a Questionnaire Page

distinction, these parts are separated by a horizontal divider. Inside the descriptive part, the actual question is displayed. Further, error messages concerning the question as a whole (e.g., the answer is mandatory) reside in this part. The interactive part of a question element can again be subdivided into a toolbar section and a data entry section. To clarify, the toolbar might not always be displayed. Toolbar elements are displayed only, if additional information may be provided by the element. If, for example, a question holds further instructional information, the “instructions” button is displayed. Clicking the button reveals a popover containing the instruction text. For non-mandatory questions whose data entry type does not allow an answer to be undone by default (e.g., radio buttons), an “undo” button is displayed, allowing users to clear their previous answer (**GES4**). In the data entry section, different, interactive input elements are displayed, according to the type of the corresponding question. Using these elements, a user is

able to provide his answer to a given question.
To give users the ability to proceed to the next page of a questionnaire, each page contains a *proceed bar*. The *proceed bar* is always displayed beneath the last question of a questionnaire page. However, for pages containing not enough questions to fill the entire screen, the bar is pushed to the bottom of the page, making it easier for users to interact with it. By giving this bar a relative instead of an absolute, always visible, position, it is ensured, especially for pages exceeding the vertical screen size, that a user is exposed to each question at least one time, before trying to proceed.
The following sections present an overview over the different types of questions that exist for a questionnaire, as well as their implementation inside the user interface.

4.5.1 Choice Question Types

The first type of questions are choice questions. Choice questions share one characteristic, where a user chooses his answer based on a set of predefined options. However, the categories of choice questions differ in the number of options, which can be selected, as well as in their visual representation.

Single Choice Question

As their name implies, single choice questions only allow for selecting a single answer out of a set of available ones. Available answers are stacked on top of each other, with each row containing the answer itself on the left, as well as a selection indicator, highlighting if a certain answer is currently selected or not, on the right. Thereby, the entire row acts as a touch target, making the selection of an answer easy and accessible. While when running on *Android*, selection indicators are represented as *radio buttons* (see Figure 4.6, left), the *iOS* version relies on a small hook to indicate that an answer is selected (see Figure 4.6, right). To further highlight a selected answer, its text is colorized to differ from non-selected answers. Available answers are separated by horizontal dividers to allow proper distinction between them. Since it is from great importance that the entire answer

is visible for the user, answers exceeding the maximum width of a row are continued in a new line. In such cases, the selection indicator is re-positioned to remain vertically centered.

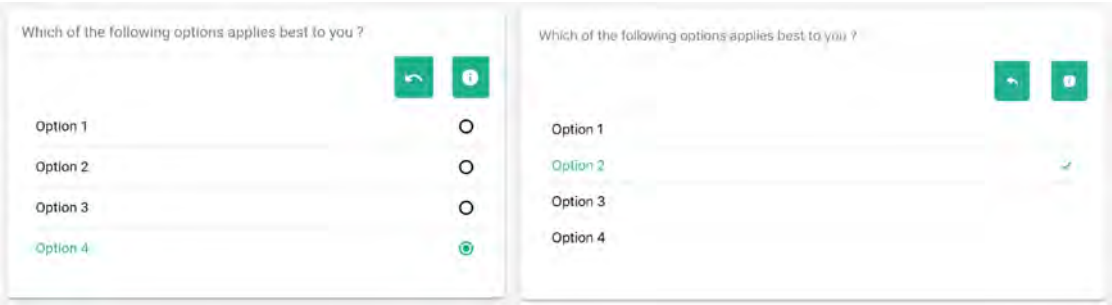

Figure 4.6: Single Choice Question for *Android* (left) and *iOS* (right)

Multiple Choice Question

Compared to single choice questions, which are restricted to a single answer, multiple choice questions allow for multiple answers to be selected. Still, their visual representation remains nearly identical. The only visual differences are in terms of appearance and placement of their respective selection indicators. Multiple choice questions make use of *checkboxes*, placed on the left side of a corresponding answer. The appearance of the latter thereby differs, according to the underlying mobile OS, which can be seen in Figure 4.7. An answer can be selected by tapping the corresponding row. Vice versa, tapping an already selected answer, deselects the latter.

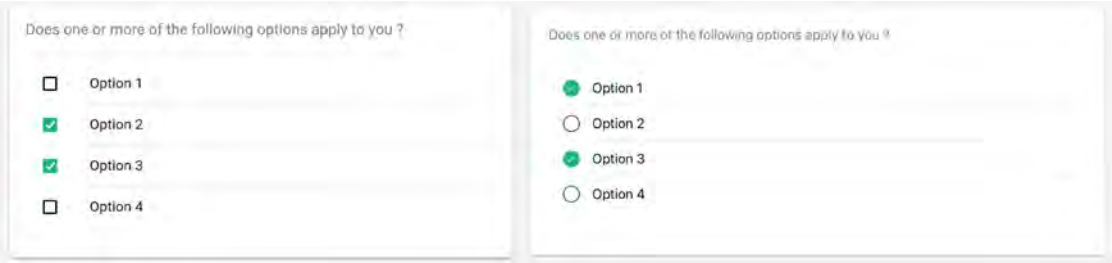

Figure 4.7: Multiple Choice Question for *Android* (left) and *iOS* (right)

Dropdown Question

While the general automatism for dropdown questions resembles the one of single choice questions, they differ in their visual representation and interaction pattern when selecting an answer. For dropdown questions, the available answers are not visible at a glance. In turn, a selection field, asking the user to choose an option is presented. By tapping the selection field, a centered popover, overlaying the actual content is displayed (see Figure 4.8). Unlike single choice questions, which select the tapped answer instantaneously, answers in dropdown questions are only visually marked as selected and require user confirmation via the “OK”–button in order to be actually selected. Alternatively, the selection can be aborted by tapping the “CANCEL”–button. To avoid a loss of context, as the actual question is likely to be hidden by the popover, the question text is displayed in the popover title.

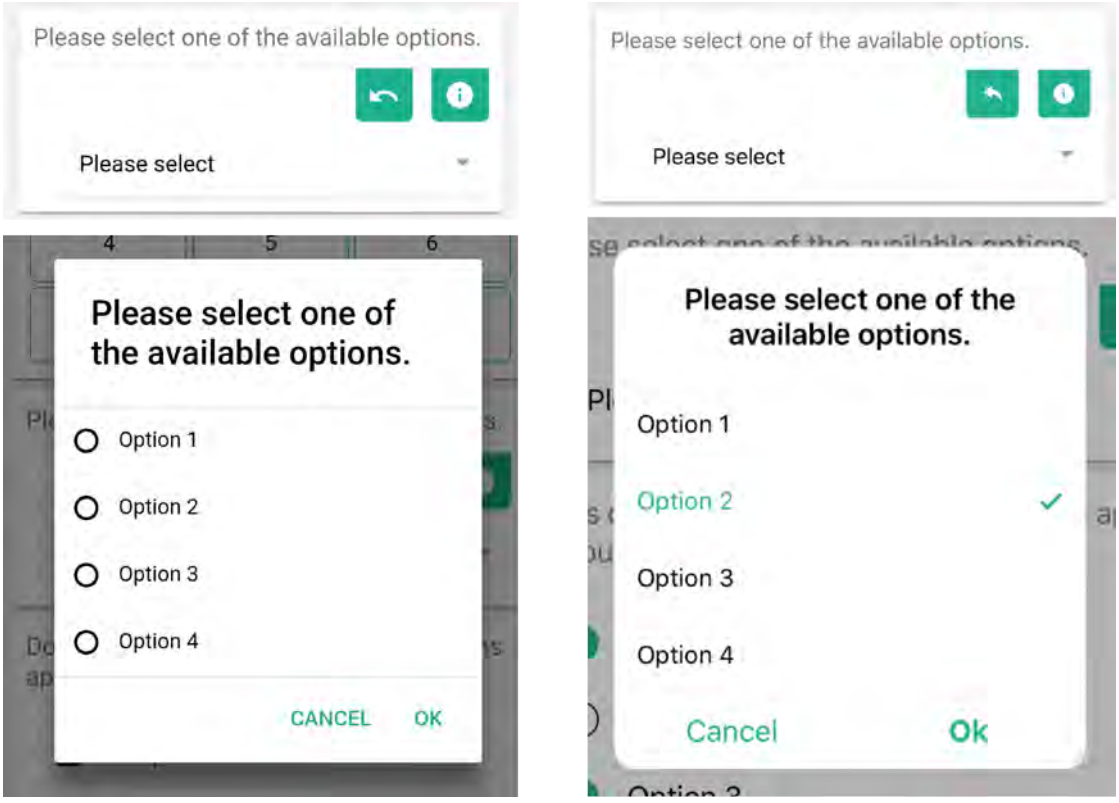

Figure 4.8: Dropdown Question for *Android* (left) and *iOS* (right)

Buttongrid Question

Buttongrid questions present a visual alternative to multiple choice questions. Instead of stacking answers on top of each other, the latter are arranged in a grid of buttons. To communicate interactivity, each answer is surrounded by a colorized border. To allow proper distinction between selected and deselected answers, color is applied to the background of selected answers and the text color becomes inverse to deselected answers. Since this type of question does not make use of platform specific components, there exist no visual differences between button grid questions on *Android* and *iOS*. However, differences in terms of layout exist for varying screen sizes (see Figure 4.9). While on tablets, each grid row can hold a maximum of five answers, this number is reduced to three, on smart phones, to provide adequate sized touch targets. Nonetheless, this type of question might not be suitable for longer answers, as the available space for each answer is fairly limited.

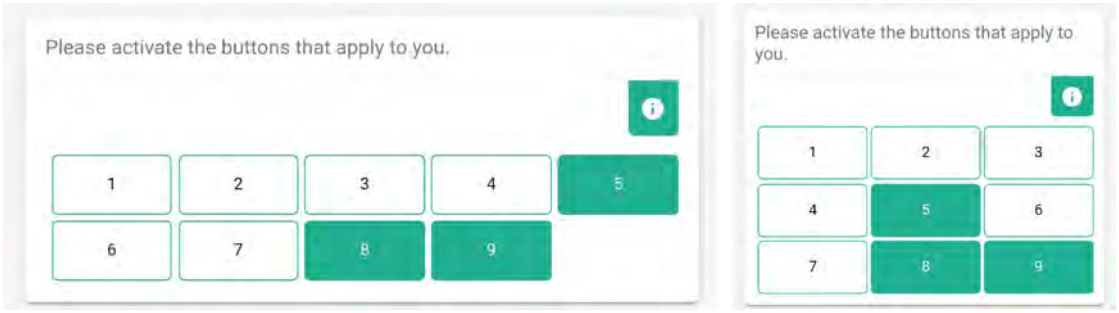

Figure 4.9: Buttongrid Question on Tablet vs. Smart Phone

4.5.2 Free Input Question Types

Free input questions allow users to provide answers themselves instead of choosing from predefined options. Therefore, most of them rely on a virtual keyboard to enter data. As the latter might not comply with **DAT1**, it has to be clarified that the person configuring a questionnaire is responsible to select appropriate question types and must not necessarily make use of free input questions. However, it has been tried to make data entry tasks requiring a keyboard as comfortable for the user as possible, since

they are unavoidable in some cases. Regarding **DAT5**, if defined in the questionnaire model, placeholders and units (e.g.,for numerical inputs) are applied to communicate the purpose and context of a certain field. To make them stand out among other questions (**DAT4**), especially if no placeholder or unit is given, a transparent rectangular fill is applied, making it fairly easy for users to identify them. Further, as shown in Figure 4.10, the current state of the input field is always communicated towards the user (**DAT6**). By tapping an input field, it becomes active, which is indicated by a colorized border and the cursor is set inside the field. Furthermore, the input field gets shifted above the expanding keyboard, enabling users to always be aware of what they are currently typing. User input is validated dynamically, according to constraints imposed by the creator of the questionnaire. If a user input, at any point, does not match a certain constraint, the input field goes into an invalid state. Thereby, the border color changes and a message, communicating the reason of invalidity, is displayed underneath the input field.

Text

For common text input, two different question types exist, with one of them making use of a regular, single-line text input field (see Figure 4.10) and the other one using a multi-line, text area. According to the expected input length, either the first or the second one can be chosen. Constraints for these questions may be a minimum or maximum number of input characters.

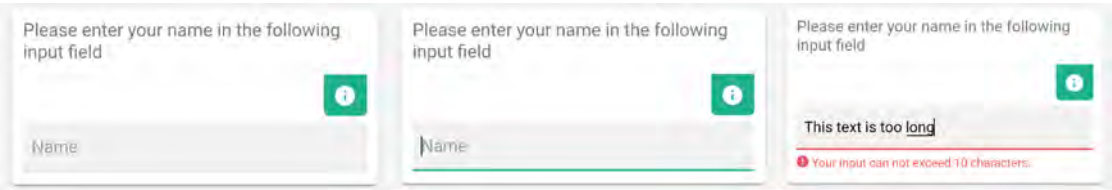

Figure 4.10: Input Field States for Text Input

Numbers

For the input of numbers, the application provides two question types, which allow for input of either integers or floating point numbers. While the visual representation of the latter shows almost no differences to questions types for textual input, question types for numerical input can additionally hold a unit, emphasizing the meaning of an input. The unit is displayed on the right side of the corresponding input (see Figure 4.11). To comply with **DAT5**, data entry tasks for these fields are done using a platform-specific numerical keyboard layout. Possible constraints for user inputs may be a specific minimum or maximum value.

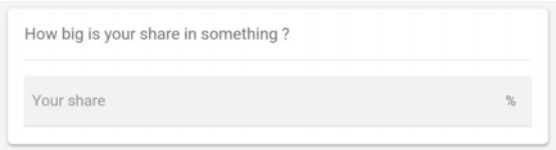

Figure 4.11: Numerical Input Question

Dates

As the input of dates using regular text fields tends to be error-prone and might be tedious to perform for users, this question type uses an alternative input method. Date entry tasks rely on platform-specific picker widgets, which overlay the currently active user interface. The respective values for day, month and year can be selected by vertically scrolling through a list of available values. The currently selected date is thereby colorized, when running on *Android* (see Figure 4.12, left), and enclosed by horizontal boarders, when running on *iOS* (see Figure 4.12, right). Buttons to confirm the selection or abort the input task are placed on top of the widget.

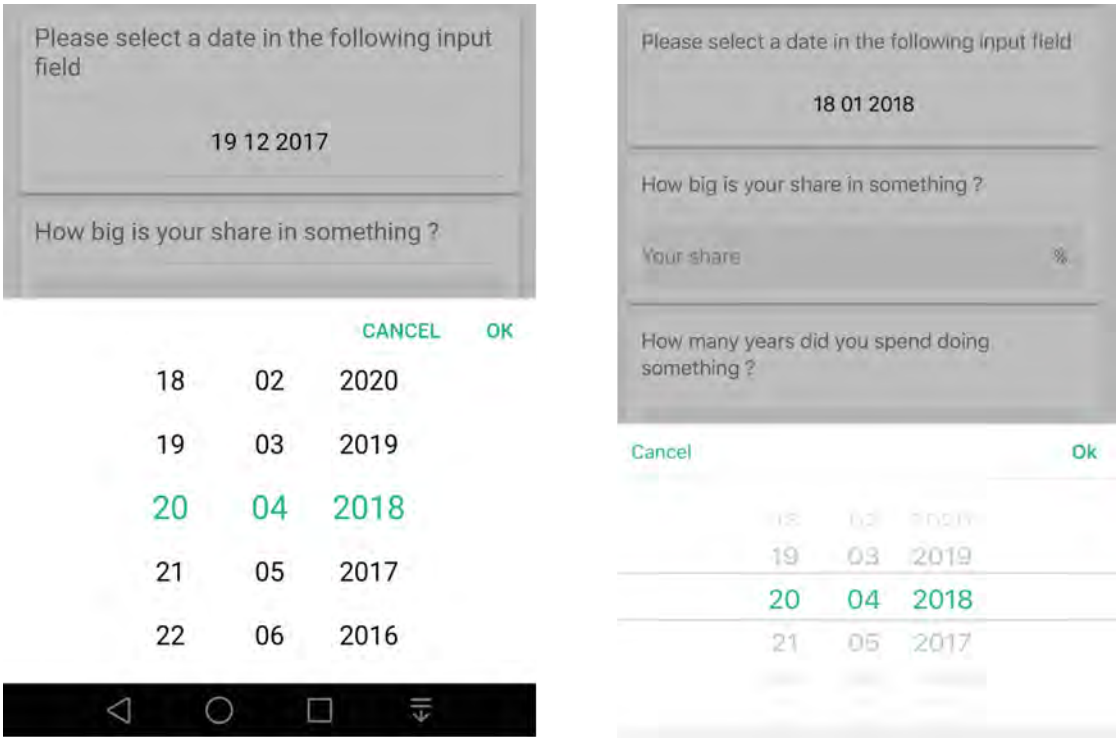

Figure 4.12: Date Input for *Android* (left) and *iOS* (right)

4.5.3 Ranking Question Types

Ranking questions, in general, enable users to bring a list of options in a specific order according to their personal preference. The application provides two variants, which slightly differ in their ranking mechanism and through that in their visual representation.

Ranking Question

This type of question displays answers as a list of items. Each item consists of a number on the left, representing the current rank of an answer, the actual answer text and a platform-specific “reorder”-icon on the right. Thereby, answers are displayed in a hierarchy, on top of each other, depicting the current ranking, with the highest ranked answer on top and the lowest ranked answer on the bottom of the list (cf. Figure 4.13). In order to change the order of a certain answer, the item can be dragged across the list and then dropped again in the desired place. On drop, the ranking number for each

answer is recalculated to match its current position in the list. To further indicate the possibility of performing drag gestures on items, additional to the “reorder”-icon, shadows are applied to each item. By this means, the item appears to lie free rather than being tied to the background.

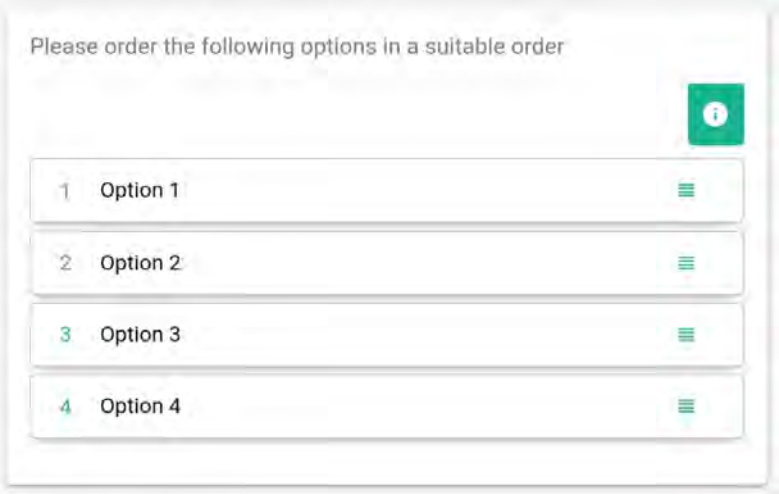

Figure 4.13: Ranking Question

Distribution Question

Different to ranking questions, which rank answers by order from 1 to n, the ranking in distribution questions is done by distributing a certain number of points between possible answers. In the course of this, the higher the number of points an answer obtains, the higher it is ranked. The distribution of points, thereby, is realized with an inline numerical input field on the right side of the corresponding answer (see Figure 4.14). Since inputs are restricted to numerical values, a numerical keyboard layout is used for data entry. In order to save calculation effort for the user, each input field possesses a placeholder, displaying the maximum number of points a user is currently able to assign to this answer. During user input, these placeholders are dynamically recalculated.

Figure 4.14: Distribution Question

4.5.4 Slider Question Types

For numerical inputs, instead of relying on keyboard input (as described in Section 4.5.2), one can make use of *Slider Questions*. In the course of this, data entry is done by dragging either one or two “knobs” on a horizontal number line, which is limited by given minimum and maximum values, displayed on the left and right side of the line (see Figure 4.15 and Figure 4.16). While dragging a “knob”, a small label, representing the current position on the number line, is displayed above it. Since the latter disappears, when releasing the “knob”, *Slider Questions* hold an always visible label, displaying the current position of the “knob”, to enable users to see their answer at any time. *Slider Questions* allow for either the entry of a single value or the entry of a number range.

Single Slider Question

To obtain a single numerical value from the user, one can use a *Single Slider Question*. The latter allows to move a single “knob” upon the number line in order to choose an answer value between the given minimum and maximum values. Thereby, dragging the “knob” to the left decreases the value and the other way round. In order to highlight the user’s selection, the segment of the number line, ranging from the minimum value up until the selected value, is colorized.

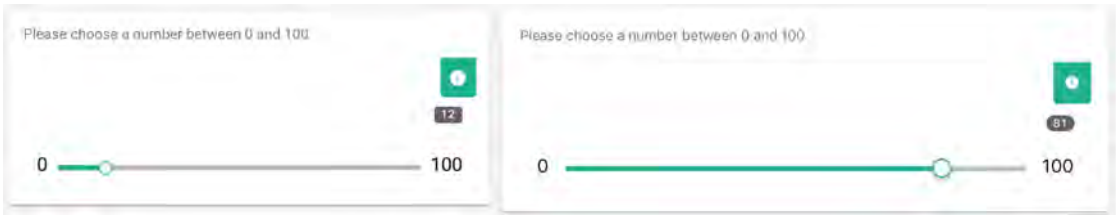

Figure 4.15: Slider Question for *Android* (left) and *iOS* (right)

Range Slider Question

Compared to *Single Slider* questions, which allow for selection of a single numerical value, *Range Slider* questions enable users to specify a range between a given minimum and maximum value. As Figure 4.16 shows, this question type uses two “knobs” in order to select a specific range. By dragging the “knobs” apart from each other, the answer range is expanded while by dragging them towards each other the range decreases. The number line segment enclosed by the “knobs” (answer range) is highlighted in color.

Figure 4.16: Range Slider Question for *Android* (left) and *iOS* (right)

4.5.5 Matrix Question

A question type, which is often used in questionnaires, is the so called *Matrix* question. It consists of multiple rows (i.e., items) and columns (i.e., choices). The user, thereby, is asked to evaluate the row items by using the same set of available column choices for each row. The application displays *Matrix* questions as tables, with header fields containing respective row and column descriptions and data fields containing selection widgets (i.e., *radio buttons*), enabling users to select an answer out of the column choices for each corresponding row. Since only one answer for each row is allowed, the selection mechanism is equal to the mechanism of *Single Choice* questions (described in

Section 4.5.1), using *radio buttons* on *Android* and tiny hooks on *iOS* (see Figure 4.17). As the number of available column choices can vary, there might be too many choices to fit into the display area reserved for the columns. Therefore, if the width of the columns exceeds the available display space, the latter become horizontally scrollable, which is indicated by applying slight inner shadows to the left and right border of the column container. However, the row descriptions remain static when scrolling columns in order to ensure context awareness.

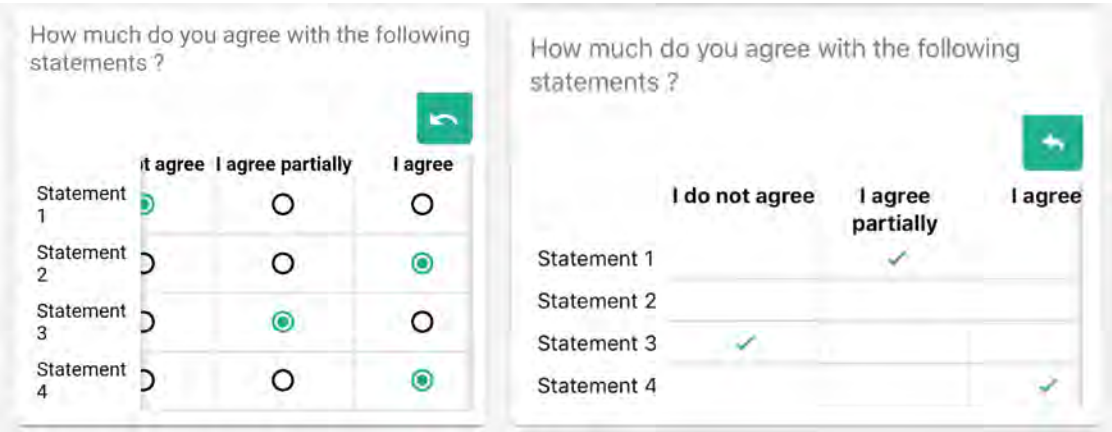

Figure 4.17: Matrix Question for *Android* (left) and *iOS* (right)

4.6 Administration Interface

The application provides a dedicated area for administration purposes. This area allows authenticated users to download new questionnaires, making the latter available in the public application area and manage questionnaire related data, such as unfinished questionnaire instances or corresponding results. Further, application settings may be configured in this area.

4.6.1 Navigation and Menu

As already described in Section 4.3, the underlying structure of the application is rather flat than deep, organizing application functionality into logical groups with each group

having its own *top-level view*. To navigate between *top-level views* in flat hierarchies, suitable menu options might be a *navigation drawer* or a *tab bar*. Since the application is mainly designed to run on tablets, which, due to their display size, can tolerate the loss of some screen space, it was chosen to use an always visible *tab bar*, on the bottom of the screen, for top-level navigation. In this way, users are able to navigate efficiently (**NAV1**), as it only takes a single tap to switch between views. Further, using a *tab bar* contributes to **NAV3**, since the current location inside the application is communicated towards the user (e.g., by highlighting the respective *tab bar* icon in color). Considering **NAV3** and **NAV4**, *sub level views* are titled properly and equiped with a “Back/Up” button to remain location awareness as well as to indicate the possibility to return to a previous view. In order to familiarize users with the navigation structure, animations are applied when transitioning between views. By this means, when navigating between *top level views*, horizontal page transition animations are applied. In turn, when navigating to *sub level views* or vice versa, the application makes use of vertical page transition animations.

4.6.2 Managing Questionnaires

The first view for users entering the administration area is the questionnaire overview. This view is divided into two segments, for *local* and *available* questionnaires (see Figure 4.18). Inside the *local* segment (Figure 4.18, left), the questionnaires, which are currently stored locally on the device, are displayed. The latter are ordered chronologically according to their download date, with the most recently downloaded questionnaire residing on top. Questionnaires, thereby, are displayed as *cards*. This allows for a clear and organized presentation of content, including information about the questionnaire itself as well as main actions associated with a certain questionnaire. Each *card*, in turn, consists of a *header*, a content area and a *footer*. The *header* contains the title of the questionnaire as well as a *favourite* button (*star*-icon) in the top-right corner. By activating this button, a questionnaire can be added to the list of public questionnaires, which are displayed in the public area of the application. The current state of a certain questionnaire is indicated by the color of the corresponding icon. By default, questionnaires are private, which is indicated by an outlined icon.

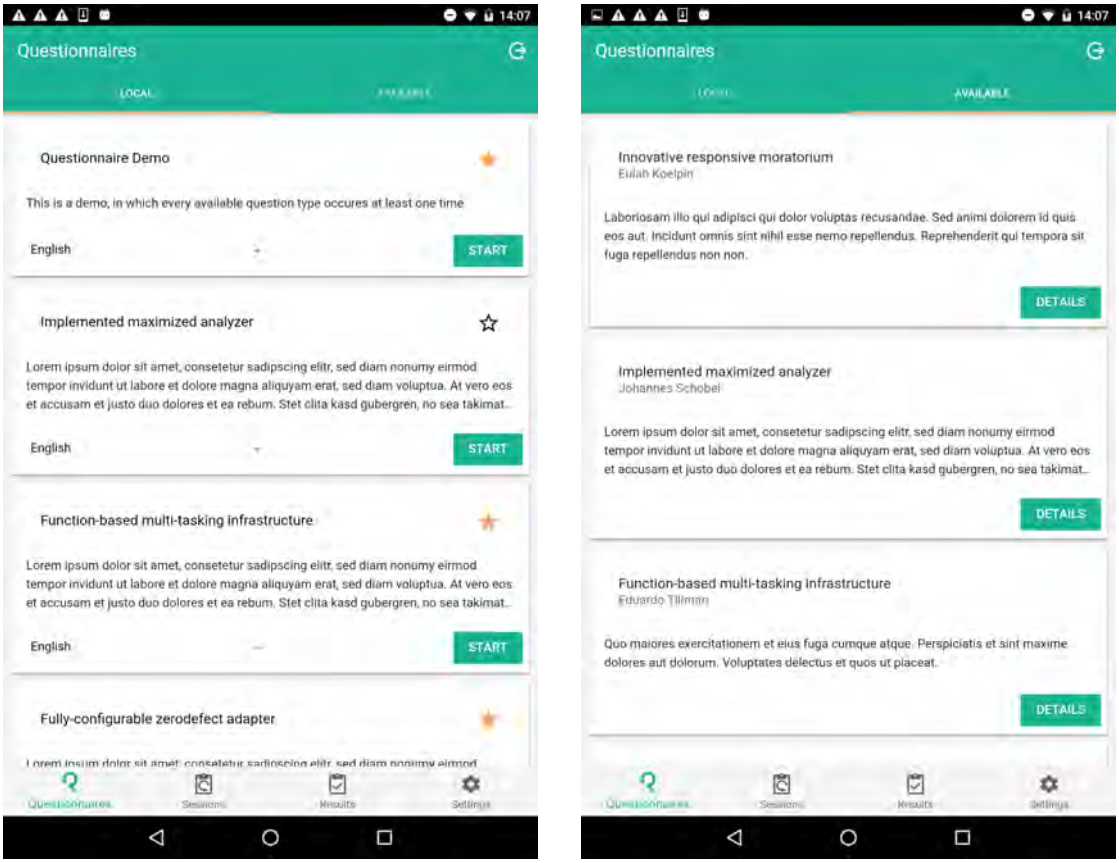

Figure 4.18: Overview of locally stored (left) and available (right) questionnaires

Inside the content area, the description of respective questionnaires is displayed. As the latter might be a longer text, its display is limited to a maximum of three lines, avoiding a questionnaire to take the entire screen space. In order to read the entire description and, furthermore, get additional information (e.g., version or contact information) to the questionnaire, a tap on the card navigates to a page displaying detailed information about the questionnaire.

To start processing a questionnaire, the “START” button in the *card footer* has to be tapped. Beforehand, the language for this execution can be changed via a selection menu besides the button, which holds an option for each language a specific questionnaire provides. By switching the language, the questionnaire title and description are automatically changed to match the chosen language.

If the device is connected to the Internet, one can find questionnaires assigned via the

QuestionSys platform inside the “AVAILABLE” segment (Figure 4.18, right). In order to view detailed information (e.g., available questionnaire versions) or download a specific questionnaire version to the device, a tap on the “DETAILS” button reveals a view, which allows for the latter.
A user can switch between the above mentioned segments by activating their respective button inside the *header bar*.

4.6.3 Managing Questionnaire Instances

Due to the potentially time-consuming questionnaires, users might decide not to fill a questionnaire in a single sitting. Hence, one should be able to pause questionnaires and resume them at a later point in time. Therefore, the application provides a dedicated view, which allows for managing (i.e., resuming) paused questionnaires. As can be seen in Figure 4.19, the paused questionnaire instances are displayed as *cards*, grouped by the corresponding questionnaire.
The *card header* displays the questionnaire title, the amount of instances associated with the questionnaire as well as a button, which reveals a hidden menu. The options in this menu allow users to directly navigate to results of the questionnaire or to remove all questionnaire instances from the device.
Beneath, the corresponding instances are listed, containing an identifier of the instance and additionally a timestamp, referring to the latest point of execution. In order to save screen space, a maximum of three instance items are visible at a peek. However, if the number of instances per questionnaire exceeds that maximum, the card can be expanded by tapping the *arrow* in the *card footer* in order to display the entire list (c.f., Figure 4.19, first item).
Inside each list, instances are ordered according to their timestamp from most recently executed to older instances. By tapping an item, one can resume this specific questionnaire instance. Sliding an element to the left, however, reveals the option to delete the targeted questionnaire instance.

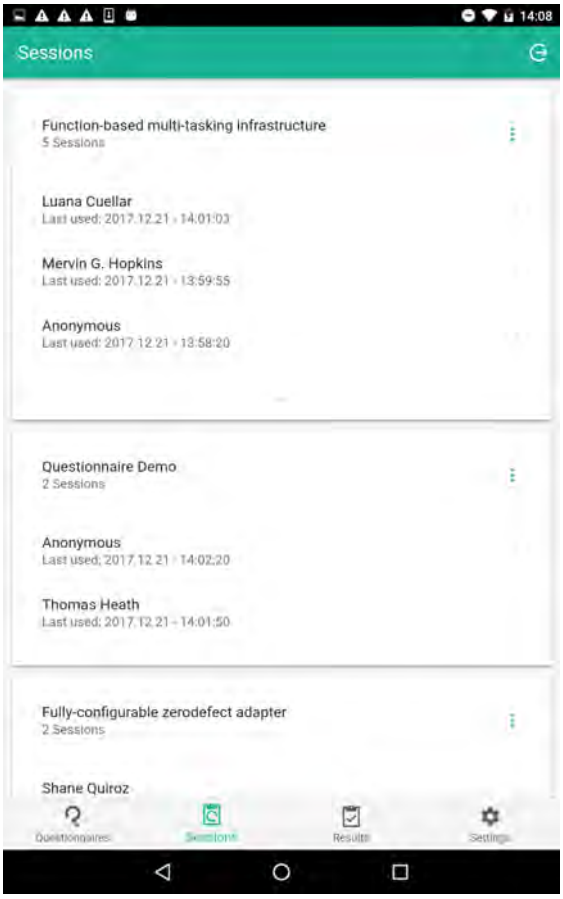

Figure 4.19: View for managing Questionnaire Instances

4.6.4 Managing Results

Besides the execution of questionnaires, another important functionality may be the management of collected results. As the number of collected results might be significantly higher than the one of available questionnaires or instances, it was decided to go with a hierarchical, list-based layout for managing results instead of relying on *cards*. The three levels of hierarchy can be seen in Figure 4.20.

In order to manage results, the entry point is a screen displaying a list of all local available questionnaires (Figure 4.20, left). Each list item contains the respective questionnaire's title, followed by the number of collected results associated with it. In this view, results are managed on a questionnaire level, meaning that actions performed on the items

affect every result associated with the corresponding questionnaire. Thereby, actions may be the upload or deletion of results of selected questionnaires. In order to be able to select questionnaires, one can either perform a *long press* gesture on a list item or, alternatively, activate the *selection* button inside the header bar to enter *selection mode* (**GES3**). Once entered, the header bar is replaced by a contextual toolbar, providing shortcuts for performable actions. Further, each list item contains a checkbox on the left, to indicate if a questionnaire is currently selected or not.

To navigate to the overview of a specific questionnaire's results (Figure 4.20, middle), its item has to be tapped. Similar to the parent view, this screen displays results as a list of items in ascending order, according to the date of collection. In this context, every result holds the corresponding participant's identifier as a title, followed by its collection date. Underneath the latter, space for colorized markers, which can be set by the interviewer in order to add context (e.g., participant needs to be interviewed again) to certain results, is left out. The right end of the item is, where status icons (e.g., result is already uploaded or result contains errors) reside. Furthermore, for already uploaded results, the background is grayed out in order to be properly distinguishable from unsynchronized results.

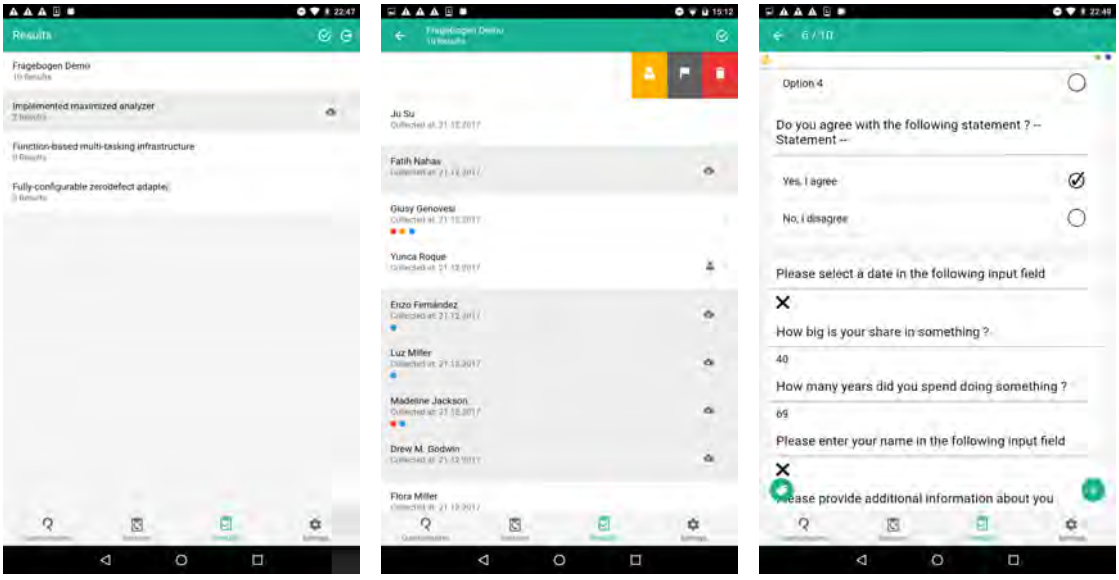

Figure 4.20: Managing results

In addition to the actions, which can be performed when enabling the selection mode (equal mechanisms as parent view but on a result level rather than the questionnaire level), buttons for sophisticated functionality, such as flagging results as *test*, applying color markers or deleting a single result, can be revealed by swiping the corresponding item to the left.
In order to reach the deepest level of hierarchy (Figure 4.20, right), namely the actual resulting answers to a questionnaire, the belonging result item needs to be tapped. To avoid having to navigate back in order to view another participants answers, one can make use of a horizontal swipe gesture. By swiping rightwards, the next recently collected result is shown, whereas by swiping leftwards, the next oldest collected result is displayed. An increasing and decreasing index inside the header bar, thereby, clearly communicates the currently displayed result. Besides viewing results, users are able to perform result related actions, such as changing the language of displayed content, send a result as *HTML* document via e-mail or marking the result, using the *floating action buttons*, which are attached to the bottom left and right corners of the screen.

4.6.5 Configure Application Settings

Like most other applications, the application developed in the context of this thesis allows authenticated users to adjust application settings to fit their specific needs, by providing a *Settings* menu. Options to be adapted may include changing the overall application appearance (e.g., changing the color scheme or the application language). Further, more sophisticated adjustments like allowing unauthenticated users to resume to questionnaire instances or gathering debug information during questionnaire execution can be made. In order to remain clarity, settings are logically grouped into system settings and settings associated with the execution of questionnaires. The settings page is shown in Figure 4.21.

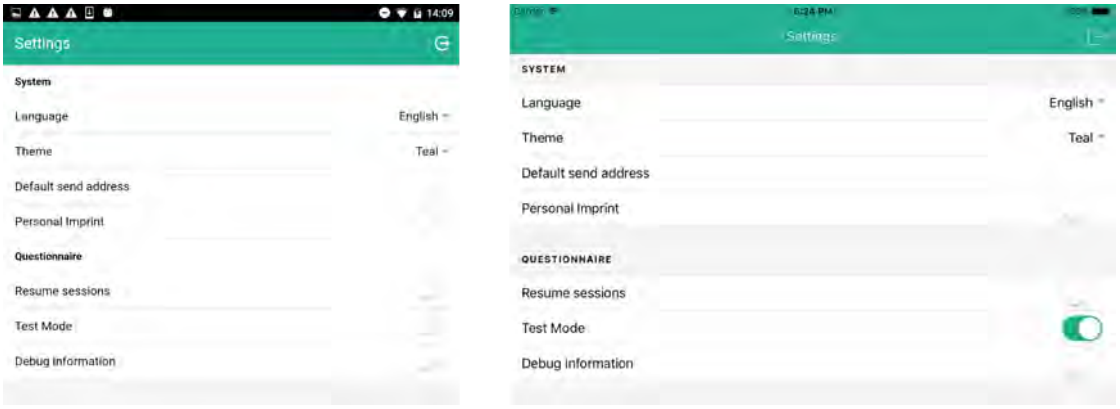

Figure 4.21: Settings Page in Administration Area for *Android* (left) and *iOS* (right)

4.7 Client Interface

Users without permission to enter the administration area (e.g., participants of a study) are still able to execute questionnaires from the public area. However, available questionnaires in this area are restricted to those, which were *pinned* by an administrator (c.f., Subsection 4.6.2). Like in the administration area, questionnaires are displayed using *cards*, though in a more reduced version. Besides the questionnaire title and description, which is restricted to three lines, *cards* only display a *dropdown menu* for language selection and “START” button to begin questionnaire execution. A tap on the *card*, thereby, expands the latter in order to display the entire questionnaire description. In turn, a tap on an expanded card contracts it again.

As participants might not be familiar with the application’s language, the latter can be changed via a *dropdown menu*, which is located inside the header bar. By changing the application language, not only application related text labels (e.g., button labels) but also the title and description of displayed questionnaires (if available in the selected language) change to match the selected language.

Through a menu, which appears upon interaction with a button inside the header bar, unauthenticated users are able to navigate to the application’s imprint, whereas administrators can additionally log into the administration area.

4.8 Implementation

The application, which interface was described in the previous sections, was developed as *hybrid application*, in order to benefit from advantages a cross-platform development approach offers (as described in Section 2.2). In the process, it was made use of the *Ionic* framework [44], an open-source web framework for *hybrid application* development, based on *Angular* and *Apache Cordova*. As imposed by *Angular*, the application is written using *HTML*, *SASS*, a *CSS* pre-processor, and *TypeScript*, a superset of *JavaScript*, which goes along with concepts such as static types and class inheritance. In this context, a component-based approach is pursued, where each part of the application is implemented as a component, which itself consists of a set of smaller components again. Each component is, thereby, defined by its own interface and execution and validation logic, making components easily interchangeable and adding modularity to the whole application. Especially for the questionnaire scenario, the component-based approach contributes benefits, as, for example, each question type can be defined as a self-executing component with its own unique interface and execution logic.
The *Ionic* framework itself provides a predefined set of *Angular* components [45], which were applied throughout the application. These components (e.g., pages, list items, cards, icons or input widgets) mimic the appearance and functionality of their native equivalents according to the OS the application is currently running on. By making use of *Ionic* components, a native look and feel may be guaranteed.
In order to gain access to OS functionality (e.g., camera, sensors or databases), one can make use of *Ionic Native Plugins* [46], which provide *TypeScript* wrapper interfaces for *Apache Cordova Plugins*. The latter, in turn, allow to invoke native routines and services, via *JavaScript* method calls.

5

Summary

This chapter briefly summarizes the relevant findings of this thesis. Further, an outlook on how future work, built on the findings and implementations in the course of this thesis, could look like.

In Chapter 2, fundamental aspects regarding usability criteria in software systems (Section 2.1) and cross-platform mobile development (Section 2.2), with special focus on hybrid mobile applications, were introduced.

Subsequently, specific rules and user interface guidelines for mobile devices (e.g., smart phones and tablets) were collected, presented and discussed in Chapter 3. In the course of this, the style guides and usability guidelines of the two major mobile platforms, namely *iOS* and *Android*, as well as work related with user interface design and development, were taken into consideration. Aspects ranging from visual design (Section 3.1), over interaction design (Section 3.2), up to navigation in mobile applications (Section 3.3) were covered in this chapter.

Chapter 4 deals with the practical part of this thesis, more specific, the development of a user interface for mobile data collection purposes that follows the described guidelines. In order to gain insight into the requirements such an user interface or, in general, such an application, needs to satisfy, potential use case scenarios are circumscribed in Section 4.1. The actual requirements derived from these use cases were listed in Section 4.2.

The latter is followed by a detailed description of the user interface developed in the context of this thesis, which goes alongside with screenshots of the resulting mobile application. Furthermore it is described how specific guidelines from Chapter 3 and requirements from Section 4.2 were applied and realized inside the user interface.

5.1 Outlook

As by now, the application developed in the context of this thesis is still in an early state. The question types that exist to the present only cover a set of basic question types, which are already known from paper-based questionnaires. At this stage, it might be useful to conduct first tests in real-world scenarios, in order to evaluate, whether or not the design decisions made actually work out the intended way. Insights on how such tests for data collection applications could look like, are described in [47].
Future work may deal with the development of more sophisticated question types, taking advantage of the possibilities mobile devices provide. Additional question types might include different media types (e.g., images, audio or video) or make use of the mobile device's built-in sensors and interfaces, for example to gather data from external devices (e.g., blood pressure monitor) via *bluetooth*. For such question types, suitable visual representations and interaction methods have to be elaborated in order to design the data collection process to be as easy as possible for end-users working with the application. Further, additional features, such as an instant evaluation of finished questionnaires according to constraints defined by the designer of the questionnaire, could be added to the application. This would allow participants, for example of a medical study, to have an instant feedback about their health status based on given answers.
However, the most important aspect is to convince people and organizations (e.g., research institutes), who currently rely on paper-based questionnaires to conduct studies, from the numerous benefits of the digital data collection approach, not only for themselves, but also for participants of their studies.

Bibliography

[1] Schobel, J., Schickler, M., Pryss, R., Reichert, M.: Process-Driven Data Collection with Smart Mobile Devices. In: 10th International Conference on Web Information Systems and Technologies (Revised Selected Papers). Number 226 in LNBIP. Springer (2015) 347–362

[2] Schobel, J., Pryss, R., Schlee, W., Probst, T., Gebhardt, D., Schickler, M., Reichert, M.: Development of Mobile Data Collection Applications by Domain Experts: Experimental Results from a Usability Study. In: 29th International Conference on Advanced Information Systems Engineering (CAiSE 2017). Number 10253 in LNCS, Springer (2017) 60–75

[3] Schobel, J., Pryss, R., Reichert, M.: Using Smart Mobile Devices for Collecting Structured Data in Clinical Trials: Results From a Large-Scale Case Study. In: 28th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2015), IEEE Computer Society Press (2015) 13–18

[4] Schobel, J., Pryss, R., Schickler, M., Ruf-Leuschner, M., Elbert, T., Reichert, M.: End-User Programming of Mobile Services: Empowering Domain Experts to Implement Mobile Data Collection Applications. In: 5th IEEE International Conference on Mobile Services (MS 2016), IEEE Computer Society Press (2016) 1–8

[5] Nielsen, J.: Usability 101: Introduction to Usability. https://www.nngroup.com/articles/usability-101-introduction-to-usability/ (2012) Accessed: 2017-10-13.

[6] Gartner: Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2017. (https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/) Accessed: 2017-11-07.

[7] Strategy Analytics: Tablet operating systems market share worldwide from 1Q'16 to 2Q'17. (https://www.statista.com/statistics/273840/global-market-share-of-tablet-operating-systems-since-2010/) Accessed: 2017-11-07.

[8] Palmieri, M., Singh, I., Cicchetti, A.: Comparison of cross-platform mobile development tools. In: 16th International Conference on Intelligence in Next Generation Networks (ICIN), IEEE (2012) 179–186

[9] Xanthopoulos, S., Xinogalos, S.: A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. In: Proceedings of the 6th Balkan Conference in Informatics, ACM (2013) 213–220

[10] Apache Cordova: Architectural overview of Cordova platform - Apache Cordova. (https://cordova.apache.org/docs/en/latest/guide/overview/index.html#architecture) Accessed: 2017-10-15.

[11] Babich, N.: The Underestimated Power Of Color In Mobile App Design — Smashing Magazine. https://www.smashingmagazine.com/2017/01/underestimated-power-color-mobile-app-design/ (2017) Accessed: 2017-10-16.

[12] Apple Inc.: Color - Visual Design - iOS Human Interface Guidelines. (https://developer.apple.com/ios/human-interface-guidelines/visual-design/color/) Accessed: 2017-10-17.

[13] material.io: Color - Style - Material Design. (https://material.io/guidelines/style/color.html) Accessed: 2017-10-17.

[14] Leavitt, M.O., Shneiderman, B.: Research-Based Web Design & Usability Guidelines. (2006)

[15] AgeLightLCC: Interface design guidelines for users of all ages. http://www.agelight.com/webdocs/designguide.pdf (2001)

[16] Holt, B.: Creating senior-friendly web sites. Issue brief (Center for Medicare Education) **1** (2000) 1–8

[17] w3.org: Understanding Success Criterion 1.4.3 | Understanding WCAG 2.0. (https://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-contrast.html) Accessed: 2017-10-17.

[18] material.io: Accessibility - Usability - Material Design. (https://material.io/guidelines/usability/accessibility.html#accessibility-color-contrast) Accessed: 2017-11-20.

[19] Hoober, S., Berkman, E.: Designing Mobile Interfaces. 1st ed edn. O'Reilly Media, Sebastopol, CA (2012)

[20] Apple Inc.: Typography - Visual Design - iOS Human Interface Guidelines. (https://developer.apple.com/ios/human-interface-guidelines/visual-design/typography/) Accessed: 2017-10-21.

[21] material.io: Typography - Style - Material Design. (https://material.io/guidelines/style/typography.html#) Accessed: 2017-10-21.

[22] Harley, A.: Icon Usability. https://www.nngroup.com/articles/icon-usability/ (2014) Accessed: 2017-10-18.

[23] Galitz, W.O.: The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques. 3rd ed edn. Wiley Pub., Indianapolis, IN (c2007)

[24] material.io: Icons - Style - Material Design. (https://material.io/guidelines/style/icons.html) Accessed: 2017-11-10.

[25] Apple Inc.: System Icons - Icons and Images - iOS Human Interface Guidelines. (https://developer.apple.com/ios/human-interface-guidelines/icons-and-images/system-icons/) Accessed: 2017-11-13.

[26] Nielsen, J.: 10 Usability Heuristics for User Interface Design. https://www.nngroup.com/articles/ten-usability-heuristics/ (1995) Accessed: 2018 - 01 - 09.

[27] Apple Inc.: Terminology - Visual Design - iOS Human Interface Guidelines. (https://developer.apple.com/ios/human-interface-guidelines/visual-design/terminology/) Accessed: 2017-10-23.

[28] material.io: Writing - Style - Material Design. (https://material.io/guidelines/style/writing.html#) Accessed: 2017-10-23.

[29] Apple Inc.: Gestures - User Interaction - iOS Human Interface Guidelines. (https://developer.apple.com/ios/human-interface-guidelines/user-interaction/gestures/) Accessed: 2017-11-18.

[30] material.io: Gestures - Patterns - Material Design. (https://material.io/guidelines/patterns/gestures.html) Accessed: 2017-11-24.

[31] Nielsen, J.: Tablet Usability. https://www.nngroup.com/articles/tablet-usability/ (2013) Accessed: 2018-01-15.

[32] Kim, J.H., Aulck, L., Thamsuwan, O., Bartha, M.C., Johnson, P.W.: The Effects of Virtual Keyboard Key Sizes on Typing Productivity and Physical Exposures. Proceedings of the Human Factors and Ergonomics Society Annual Meeting **57** (2013) 887–891

[33] Apple Inc.: Data Entry - User Interaction - iOS Human Interface Guidelines. (https://developer.apple.com/ios/human-interface-guidelines/user-interaction/data-entry/) Accessed: 2017-12-09.

[34] Budiu, R.: A Checklist for Designing Mobile Input Fields. https://www.nngroup.com/articles/mobile-input-checklist/ (2015) Accessed: 2017-12-09.

[35] material.io: Text fields - Components - Material Design. (https://material.io/guidelines/components/text-fields.html#) Accessed: 2017-12-09.

[36] Whitenton, K.: Flat vs. Deep Website Hierarchies. https://www.nngroup.com/articles/flat-vs-deep-hierarchy/ (2013) Accessed: 2017-10-26.

[37] Apple Inc.: Navigation - App Architecture - iOS Human Interface Guidelines. (https://developer.apple.com/ios/human-interface-guidelines/app-architecture/navigation/) Accessed: 2017-11-15.

[38] material.io: Navigation - Patterns - Material Design. (https://material.io/guidelines/patterns/navigation.html) Accessed: 2017-11-15.

[39] Kurniawan, S., Zaphiris, P.: Research-derived web design guidelines for older people. In: Proceedings of the 7th international ACM SIGACCESS Conference on Computers and Accessibility, ACM (2005) 129–135

[40] developer.android.com: Navigation with Back and Up . (`https://developer.android.com/design/patterns/navigation.html`) Accessed: 2017-11-18.

[41] Reidel, A.: Entwicklung eines Designkonzepts für unterschiedliche Anwendungsszenarien eines generischen Fragebogensystems. Bachelor thesis at Ulm University (2015)

[42] Joachim, J.: Entwicklung neuer Anwendungsszenarien für ein prozessorientiertes Fragebogensystem. Masters thesis at Ulm University (2017)

[43] materialpalette.com: Material Design Colors. (`https://www.materialpalette.com/colors`) Accessed: 2018-01-14.

[44] ionicframework.com: Build Amazing Native Apps and Progressive Web Apps with Ionic Framework and Angular. (`https://ionicframework.com/framework`) Accessed: 2018-01-12.

[45] ionicframework.com: Ionic Component Documentation. (`https://ionicframework.com/docs/components/`) Accessed: 2018-01-12.

[46] ionicframework.com: Ionic Native . (`https://ionicframework.com/docs/native/`) Accessed: 2018-01-12.

[47] Bandic, J.: Evaluierung einer mobilen Anwendung zur flexiblen Datenerhebung anhand einer Benutzerstudie. Bachelor thesis at Ulm University (2017)

List of Figures

List of Tables

Name: Robin Martin Matrikelnummer: 857754

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Robin Martin